جمـهورية الجزائرية الديمقراطية الشعبيـة
وزارة التعليم العالي والبحث العلمي

جـامعـة سعـيـدة د. مـولاي الطـاهـر
كلية الرياضيات و الإعلام الآلي و الاتصالات السلكية و اللاسلكية
قسم: الإعلام الآلي

Mémoire de Master en informatique

Spécialité : Réseau Informatiques et Systémes Répartis

T h è m e

Conception et mise en œuvre d'un système de détection d'intrusion intelligent pour les réseaux IoT

▪ **Présenté par :**
Kadi El habib

▪ **Dirigé par :**
Dr. Houacine Abdelkrim

Année universitaire 2025-2026

Remerciements

*Avant tout, je remercie **Allah le Tout-Puissant** de m'avoir accordé la santé, la volonté et la patience nécessaires pour accomplir ce travail.*

*Je tiens à exprimer ma profonde reconnaissance à ma très chère **mère**, qui a toujours été mon plus grand soutien tout au long de mon parcours. Par son amour, ses sacrifices, ses encouragements et ses prières, elle m'a donné la force d'avancer et de surmonter les difficultés. Aucun mot ne pourra réellement exprimer toute ma gratitude envers elle.*

*J'adresse également mes sincères remerciements à toute ma **famille** pour leur présence, leur confiance et leur soutien moral permanent durant toutes ces années d'études.*

*Je remercie chaleureusement mes chers **amis** pour leur aide, leurs conseils, leur motivation et tous les moments partagés qui ont rendu ce parcours plus agréable et plus enrichissant.*

*Mes remerciements vont aussi à mon **encadreur** pour ses orientations, ses remarques constructives et son accompagnement durant la réalisation de ce mémoire.*

Enfin, je remercie toutes les personnes qui ont contribué, de près ou de loin, à l'aboutissement de ce travail.

Dédicace

Je dédie ce modeste travail :

*À ma très chère **mère**, pour son amour infini, ses sacrifices, sa patience et ses prières qui ont toujours illuminé mon chemin. Aucune réussite n'aurait été possible sans son soutien et sa présence à mes côtés.*

*À ma **famille**, pour leur confiance, leurs encouragements et leur soutien constant tout au long de mon parcours universitaire.*

*À mes chers **amis**, qui ont partagé avec moi les moments de doute, de fatigue et de réussite, et qui ont toujours été présents par leurs conseils et leur motivation.*

À toutes les personnes qui ont cru en moi et m'ont encouragé à avancer malgré les difficultés.

Résumer Abstract / ملخص

Résumé (Français)

La sécurité des réseaux IoT constitue un défi majeur en raison des ressources limitées des équipements connectés et de la diversité des attaques qui les ciblent. Ce mémoire présente la conception et la mise en œuvre d'un système de détection d'intrusion (IDS) intelligent, orienté live, capable de détecter plusieurs types d'attaques IoT dans un environnement simulé. La contribution principale réside dans la construction d'un pipeline de détection continu, reliant le simulateur Cooja, un mécanisme d'extraction de features par fenêtre glissante et un modèle Random Forest, le tout supervisé par un dashboard web développé avec FastAPI et WebSocket. Le modèle a été entraîné sur un dataset original construit à partir de simulations couvrant neuf comportements réseau distincts. Dix features réseau — mesurant l'envoi, la réception, les pertes, les délais et la cohérence des échanges — ont été extraites et justifiées analytiquement. La validation expérimentale, conduite sur 6 702 prédictions, établit une accuracy globale de 80,53 %, avec huit classes correctement identifiées sur neuf. Les confusions résiduelles, se concentrant désormais entre les attaques selective, forwarding et routing_blackhole, s'expliquent par la forte similarité de leurs signatures de perte de paquets dans l'espace des features retenues.

Mots-clés : *IoT, IDS, Machine Learning, Random Forest, Cooja, Détection d'intrusion, Fenêtre glissante, Réseaux de capteurs.*

Résumer

Abstract (English)

The security of IoT networks remains a significant challenge, given the resource constraints of connected devices and the growing diversity of threats targeting these environments. This thesis presents the design and implementation of an intelligent, live-oriented intrusion detection system (IDS) for simulated IoT networks. The main contribution of this work lies in the development of a continuous detection pipeline that connects the Cooja network simulator, a sliding-window feature extraction mechanism, and a Random Forest classifier, all integrated with a web dashboard built on FastAPI and WebSocket. The model was trained on an original dataset constructed from Cooja simulations covering nine distinct network behaviors. Ten network features — capturing transmission rate, reception, packet loss, response delay, and communication consistency — were extracted and analytically justified. Experimental validation on 6,702 predictions yields an overall accuracy of 80.53%, with eight out of nine classes correctly identified. Residual confusion, now primarily concentrated among selective, forwarding, and routing_blackhole attacks, is attributed to the strong similarity of their packet loss signatures in the retained feature space.

Keywords: *IoT, IDS, Machine Learning, Random Forest, Cooja, Intrusion Detection, Sliding Window, Wireless Sensor Networks.*

Résumer

الملخص (العربية)

يُعدُّ أمن شبكات إنترنت الأشياء تحدياً بالغ الأهمية، نظراً لمحدودية موارد الأجهزة المتصلة وتنوع التهديدات التي تستهدفها. يقدم هذا البحث تصميم وتنفيذ نظام ذكي للكشف عن التسلل، يعمل في الوقت الفعلي، ومخصص لشبكات إنترنت الأشياء المحاكاة.

تتمثل المساهمة الرئيسية لهذا العمل في تطوير مسار معالجة بيانات مستمر يربط بين محاكي الشبكات، وآلية لاستخراج الخصائص تعتمد على تقنية النافذة المنزلقة، ونموذج الغابة العشوائية للتصنيف. تم دمج هذا النظام مع واجهة ويب للمراقبة مبنية بتقنيات الاتصال المباشر.

تم تدريب النموذج باستخدام مجموعة بيانات أصلية تم توليدها عبر المحاكي، وتشمل تسعة سلوكيات شبكية مختلفة. وقد تم استخراج وعزل عشر خصائص شبكية تعكس معدلات الإرسال والاستقبال، وحجم فقدان الحزم، وزمن الاستجابة، وتناسق الاتصالات، مع تقديم تبرير تحليلي دقيق لاختيار كل منها.

بالمائة، مع 80.53 عينة اختبار، تحقيق دقة إجمالية بلغت 6702أظهرت نتائج التحقق التجريبي، والتي شملت نجاح النظام في التعرف الصحيح على ثمانية أصناف من أصل تسعة. أما حالات الخلط المتبقية، والتي تنحصر بشكل أساسي بين هجمات الحجب الانتقائي والتوجيه الخبيث، فتُعزى إلى التشابه الكبير في الأنماط السلوكية لفقدان الحزم الخاصة بها ضمن نطاق الخصائص المدروسة.

الكلمات المفتاحية: *إنترنت الأشياء، كشف التسلل، التعلم الآلي، Cooja Random Forest، المراقبة في ، النافذة المنزلقة، شبكات الاستشعار اللاسلكية.*

Table des matières

Liste des abréviations

AI	Artificial Intelligence
API	Application Programming Interface
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
Cooja	Simulateur de réseaux IoT de Contiki-NG
CSV	Comma-Separated Values
DoS	Denial of Service
HTTP	HyperText Transfer Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
RF	Random Forest
RPL	Routing Protocol for Low-Power and Lossy Networks
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WebSocket	Protocole de communication bidirectionnelle temps réel
WSN	Wireless Sensor Network
XML	eXtensible Markup Language

Table des figures

Liste des tableaux

Introduction Générale

Contexte général

L'Internet des Objets connaît aujourd'hui une expansion rapide dans de nombreux secteurs tels que la santé, l'industrie, l'agriculture et les villes intelligentes. Grâce à l'interconnexion d'objets capables de collecter, transmettre et exploiter des données, les réseaux IoT jouent un rôle important dans l'automatisation, la supervision et le contrôle de plusieurs environnements. Cette évolution technologique a permis d'améliorer l'efficacité de nombreux systèmes, mais elle s'accompagne également de nouveaux défis, notamment en matière de sécurité **[1], [2]**.

En effet, les réseaux IoT demeurent particulièrement vulnérables en raison de la diversité des équipements connectés, de leurs ressources souvent limitées et de leur exposition permanente aux échanges réseau. Ces caractéristiques augmentent le risque d'attaques susceptibles de perturber les communications, de compromettre les données et d'affecter le bon fonctionnement du système. Dans ce contexte, la protection des réseaux IoT devient un enjeu essentiel, ce qui justifie le recours à des mécanismes intelligents capables de surveiller l'activité du réseau et de détecter les comportements suspects [3], [4].

Problématique

Le développement rapide des réseaux IoT s'accompagne d'une augmentation des risques de sécurité liés à la diversité des équipements connectés, à leurs contraintes techniques et à leur exposition permanente aux échanges réseau. Dans ces environnements, plusieurs attaques peuvent perturber les communications, compromettre les données et affecter le bon fonctionnement du système. La sécurisation des réseaux IoT devient ainsi un enjeu majeur, d'autant plus que les mécanismes classiques de surveillance ne permettent pas toujours de détecter efficacement des comportements malveillants dans des contextes dynamiques et hétérogènes.

Dans cette perspective, le recours à un système de détection d'intrusion apparaît comme une solution pertinente pour surveiller l'activité du réseau et identifier les comportements anormaux. Toutefois, afin d'améliorer les capacités de détection face à la diversité des attaques, il devient nécessaire d'intégrer une approche fondée sur l'intelligence artificielle, capable

d'exploiter les données du réseau et d'automatiser la reconnaissance des activités suspectes **[5], [6]**.

La problématique de ce travail consiste ainsi à étudier comment renforcer la sécurité d'un réseau IoT à l'aide d'un système IDS fondé sur l'intelligence artificielle, capable de détecter automatiquement les attaques et de contribuer à la protection du système.

Objectifs du projet

L'objectif principal de ce projet est de concevoir un système de détection d'intrusion capable de renforcer la sécurité d'un réseau IoT face à différentes attaques. Il s'agit de mettre en place une solution permettant d'analyser l'activité du réseau et de détecter les comportements anormaux susceptibles de menacer son fonctionnement.

Plus spécifiquement, ce travail vise à :

- exploiter les données issues d'un environnement IoT simulé avec Cooja ;
- extraire des caractéristiques pertinentes du trafic réseau ;
- appliquer un modèle de classification fondé sur l'intelligence artificielle pour détecter automatiquement les attaques ;
- proposer une interface de supervision permettant de visualiser clairement les résultats de détection et de faciliter le suivi de l'état du réseau.

Organisation du mémoire

Ce mémoire est organisé en six chapitres. Le premier chapitre présente le contexte général et les enjeux de sécurité dans les réseaux IoT. Le deuxième chapitre est consacré à l'analyse du besoin et à l'étude de l'existant. Le troisième chapitre décrit la conception générale du système proposé. Le quatrième chapitre traite de l'analyse et du choix des features utilisées pour la détection. Le cinquième chapitre présente la mise en œuvre de la solution développée. Enfin, le sixième chapitre expose les tests effectués ainsi que les résultats obtenus. Le mémoire se termine par une conclusion générale résumant les apports du travail et les perspectives envisagées.

Chapitre 1 : Contexte générale et enjeux de sécurité dans l'IoT

1.1 Introduction

Ce chapitre a pour objectif de présenter le cadre théorique général dans lequel s'inscrit ce projet. Il introduit d'abord les notions fondamentales liées à l'Internet des Objets, en mettant l'accent sur les caractéristiques de ces environnements, leurs domaines d'application et les contraintes qui les distinguent des réseaux classiques. Il met ensuite en évidence les enjeux de sécurité qui accompagnent le développement rapide des réseaux IoT, notamment en raison de l'hétérogénéité des équipements et de leur exposition à différentes formes d'attaques.

Dans ce contexte, les systèmes de détection d'intrusion occupent une place essentielle dans la protection des réseaux IoT. Ce chapitre présente également l'intérêt des approches fondées sur l'intelligence artificielle pour améliorer la détection des comportements anormaux, ainsi que le rôle de la simulation avec Cooja comme cadre expérimental utilisé dans ce travail.

1.2 Internet des Objets

1.2.1 Définition de l'IoT

L'Internet des Objets, généralement désigné par l'acronyme IoT (Internet of Things), représente un paradigme technologique fondé sur l'interconnexion d'objets physiques capables de collecter, transmettre, recevoir et exploiter des données à travers un réseau de communication. Ces objets peuvent prendre des formes très variées, telles que des capteurs, des actionneurs, des équipements embarqués ou encore des dispositifs intelligents intégrés dans différents environnements d'usage. L'objectif principal de l'IoT est de permettre à ces entités de communiquer entre elles ou avec des systèmes de traitement afin d'assurer des fonctions de surveillance, d'automatisation, de contrôle ou d'aide à la décision**[7]**.

Cette interconnexion massive ne se limite pas à une simple extension d'Internet vers des objets physiques. Elle introduit de nouvelles problématiques liées à la gestion des données, à l'interopérabilité des équipements et surtout à la sécurité des communications. En effet, plus le nombre d'objets connectés augmente, plus la surface d'attaque du système s'élargit, ce qui rend la surveillance et la protection des réseaux IoT indispensables.

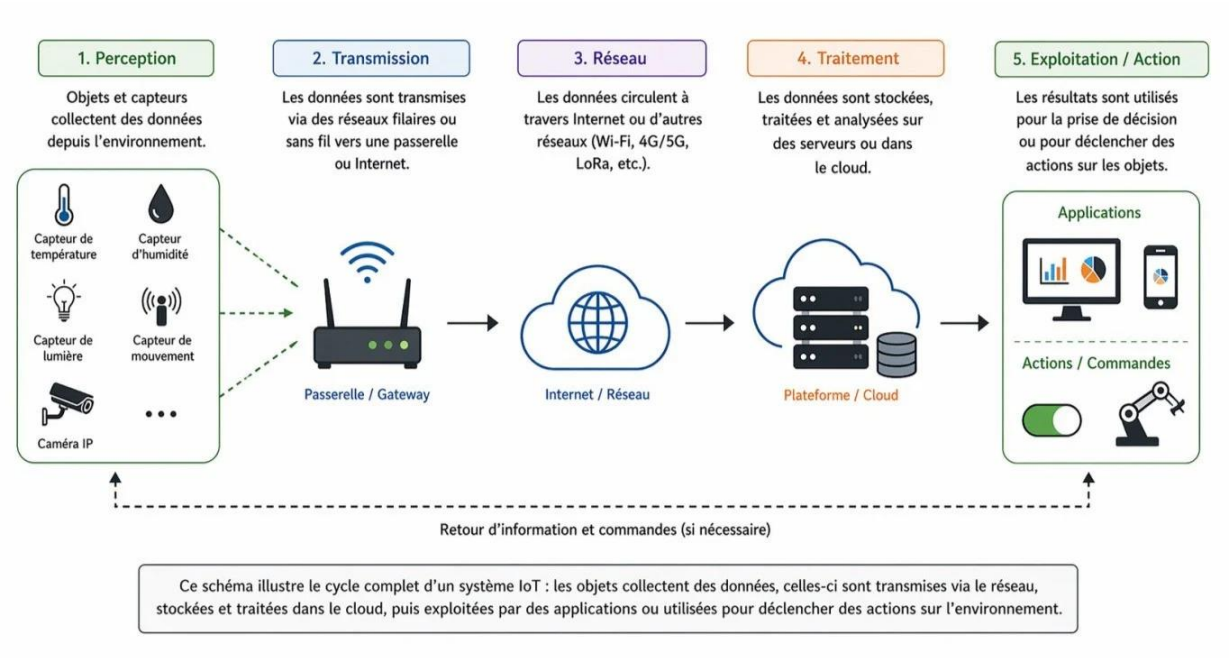


Figure 1.1 : Schéma général du fonctionnement d'un environnement IoT

1.2.2 Caractéristiques des environnements IoT

Les environnements IoT se distinguent des réseaux informatiques classiques par plusieurs caractéristiques techniques qui influencent directement leur conception et leur

sécurité. Tout d'abord, ils reposent sur un grand nombre d'objets connectés hétérogènes, dont les rôles et les capacités matérielles varient considérablement d'un équipement à l'autre.

Par ailleurs, la connectivité de ces environnements est souvent continue et dynamique. Les échanges entre les objets, les passerelles et les plateformes de supervision se font de manière régulière, parfois en temps réel, ce qui augmente la surface d'exposition aux menaces réseau. Enfin, le fort besoin d'interopérabilité entre des équipements de fabricants différents, utilisant des protocoles variés, ajoute une complexité supplémentaire à la gestion globale du système.

1.2.3 Domaines d'application de l'IoT

L'Internet des Objets est aujourd'hui utilisé dans de nombreux secteurs. Dans le domaine de la santé, il permet le suivi à distance des patients à l'aide de dispositifs connectés. Dans l'industrie, il contribue à l'automatisation des processus et à la surveillance des équipements. Il intervient également dans l'agriculture intelligente pour optimiser l'irrigation et le suivi des cultures, ainsi que dans les villes intelligentes pour gérer le trafic, l'éclairage public et la consommation énergétique.

Cette diversité d'applications montre que l'IoT occupe une place croissante dans des systèmes critiques, ce qui renforce davantage la nécessité d'assurer leur sécurité et leur surveillance.

1.3 Sécurité dans les réseaux IoT

1.3.1 Enjeux de sécurité

La sécurité constitue un enjeu majeur dans les environnements IoT en raison de la forte interconnexion entre les objets, les réseaux et les services distants. Ces systèmes manipulent souvent des données sensibles et assurent parfois des fonctions importantes de surveillance ou de contrôle. Toute atteinte à leur sécurité peut ainsi avoir des conséquences directes sur le bon fonctionnement du réseau et sur la fiabilité des services associés.

Les principaux enjeux concernent la confidentialité, l'intégrité et la disponibilité des données. Il est essentiel de protéger les informations échangées contre l'interception non autorisée, d'empêcher leur modification malveillante et de garantir un accès continu aux services**[8]**. Ces exigences justifient le recours à des mécanismes de surveillance et de détection adaptés aux contraintes propres aux environnements IoT.

1.3.2 Vulnérabilités des environnements IoT

Les environnements IoT présentent plusieurs vulnérabilités spécifiques. D'une part, de nombreux objets connectés disposent de ressources limitées en mémoire, en calcul et en énergie, ce qui réduit la possibilité d'intégrer des mécanismes de sécurité avancés. D'autre part,

l'hétérogénéité des équipements et des protocoles rend la gestion globale du système plus complexe et peut créer des points faibles dans la communication.

Par ailleurs, certains dispositifs sont déployés avec des configurations peu sécurisées ou des mécanismes de mise à jour insuffisants, ce qui augmente le risque d'exploitation par des attaquants. La combinaison de ces facteurs fait des réseaux IoT des environnements particulièrement sensibles aux menaces.

1.3.3 Principales attaques dans les réseaux IoT

Les réseaux IoT peuvent être exposés à plusieurs types d'attaques. L'attaque par déni de service (DoS) vise à saturer le réseau afin de perturber son fonctionnement normal. L'attaque blackhole consiste pour un nœud malveillant à absorber les paquets sans les retransmettre, provoquant ainsi des pertes de données. L'attaque delay introduit un retard anormal dans les communications, tandis que l'attaque replay repose sur la répétition de messages déjà transmis pour perturber le système.

D'autres attaques, telles que le selective forwarding et les attaques de routage, constituent également des menaces importantes**[10], [11]** Le selective forwarding consiste à ne retransmettre que certains paquets de manière sélective, rendant la détection difficile. Les attaques de routage, quant à elles, visent à manipuler les tables de routage afin de détourner ou bloquer le trafic réseau. Ces différentes formes d'attaques montrent la nécessité de disposer de mécanismes de détection capables d'identifier aussi bien les perturbations visibles que les comportements anormaux plus subtils.

1.4 Systèmes de détection d'intrusion pour l'IoT

1.4.1 Définition d'un IDS

Un système de détection d'intrusion, ou IDS (Intrusion Detection System), est un mécanisme de sécurité conçu pour surveiller l'activité d'un réseau ou d'un système afin d'identifier des comportements suspects ou malveillants. Son rôle principal est de détecter les anomalies, les attaques ou les tentatives d'accès non autorisé, puis de signaler ces événements à travers des alertes ou des journaux d'analyse.

Dans les environnements IoT, l'IDS joue un rôle particulièrement important, car il permet d'observer le trafic généré par les objets connectés et de repérer des comportements anormaux qui pourraient indiquer une intrusion ou une perturbation du réseau**[12]**. Il constitue ainsi un outil essentiel de surveillance et d'aide à la supervision de la sécurité.

1.4.2 Types d'IDS

Les systèmes de détection d'intrusion peuvent être classés selon leur mode de fonctionnement. Les IDS à base de signatures détectent les attaques en comparant l'activité observée à des modèles connus. Cette approche est efficace pour reconnaître des attaques déjà identifiées, mais reste limitée face à de nouvelles menaces.

Les IDS à base d'anomalies reposent sur l'analyse du comportement normal du système afin de repérer toute activité inhabituelle. Cette approche est particulièrement adaptée aux environnements IoT, où certaines attaques se manifestent par des variations de trafic, de délai ou de perte de paquets. Certains systèmes combinent les deux approches afin d'améliorer la qualité de la détection et de réduire les limites de chaque méthode prise séparément.

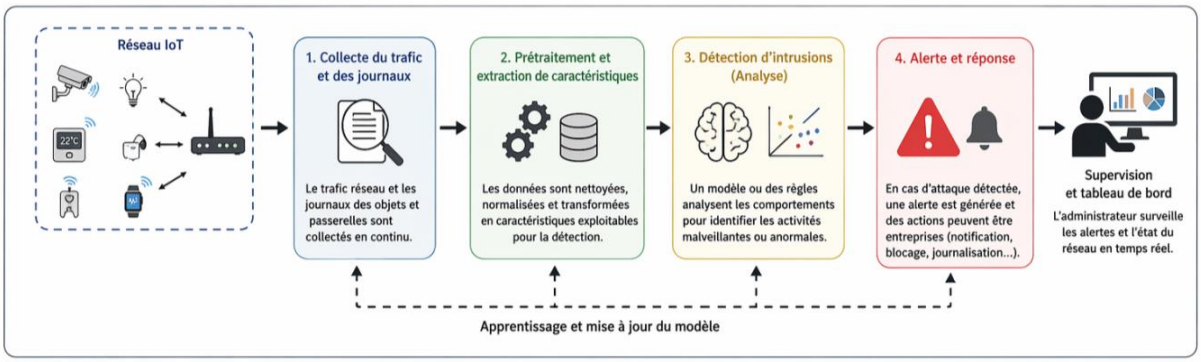
1.4.3 Intérêt de l'intelligence artificielle dans la détection d'intrusion

L'intelligence artificielle représente aujourd'hui une approche particulièrement pertinente pour améliorer les capacités des systèmes de détection d'intrusion**[13], [14]**. Dans des environnements IoT dynamiques et hétérogènes, les comportements anormaux ne se manifestent pas toujours de manière simple ou directement visible. L'utilisation de modèles de classification permet alors d'exploiter les données du réseau afin d'identifier des patterns associés à différentes attaques.

L'intérêt principal de l'IA réside dans sa capacité à apprendre à partir d'exemples et à reconnaître des relations entre plusieurs variables observées dans le trafic. En analysant des caractéristiques telles que le nombre de messages envoyés, les pertes, les délais de réponse ou la cohérence entre requêtes et réponses, un modèle intelligent peut distinguer un comportement normal d'une activité potentiellement malveillante.

Dans le cadre d'un IDS appliqué à l'IoT, l'intelligence artificielle ne remplace pas les mécanismes classiques de surveillance, mais les complète en apportant une capacité d'analyse

plus adaptée à la complexité des échanges. Elle constitue donc un levier important pour cons-



truire des systèmes de détection plus réactifs et mieux adaptés à l'évolution des menaces.

Figure 1.2 : Principe général d'un système de détection d'intrusion dans un environnement IoT

1.5 Simulation des réseaux IoT avec Cooja

1.5.1 Présentation de Cooja

Cooja est un simulateur largement utilisé dans l'étude des réseaux IoT et des réseaux de capteurs**[15], [16]**. Il permet de reproduire un environnement expérimental contrôlé dans lequel plusieurs nœuds peuvent être déployés, configurés et observés selon différents scénarios. Grâce à cet outil, il est possible d'analyser le comportement du réseau, de générer des échanges entre les nœuds et d'étudier l'impact de certaines perturbations ou attaques sans avoir recours à une infrastructure physique réelle.

1.5.2 Rôle de Cooja dans ce projet

Dans le cadre de ce projet, Cooja ne constitue pas uniquement un outil de simulation, mais aussi la source principale des données exploitées par le système de détection. Les scénarios exécutés dans l'environnement simulé génèrent des logs qui reflètent l'activité du réseau et les comportements observés entre les nœuds. Ces informations servent ensuite de base pour extraire les features nécessaires à l'analyse et à l'application du modèle de détection.

Le lien entre Cooja et le système proposé repose sur une chaîne cohérente : la simulation produit les logs, les logs sont traités pour construire les variables d'entrée du modèle, puis les résultats de détection sont transmis vers l'interface de supervision. Cette organisation montre que Cooja joue un rôle central dans la validation expérimentale de la solution développée.

1.6 Synthèse et positionnement du projet

Les éléments présentés dans ce chapitre permettent de situer clairement le projet dans son contexte scientifique et technique. Les réseaux IoT, malgré leur utilité croissante, restent exposés à de multiples attaques que les mécanismes classiques de surveillance ne permettent pas toujours de détecter efficacement. Les systèmes IDS fondés sur l'intelligence artificielle représentent une réponse adaptée à cette problématique, en combinant l'analyse automatique des données réseau et la classification des comportements suspects.

Le présent projet s'inscrit dans cette perspective en proposant un système IDS appliqué à un environnement IoT simulé avec Cooja**[17], [18]**. Il exploite les logs générés par la simulation, en extrait des variables pertinentes et applique un modèle de classification fondé sur l'IA afin de détecter automatiquement les attaques et d'en superviser les résultats en continu.

1.7 Conclusion

À travers ce chapitre, il apparaît que les environnements IoT offrent de nombreuses possibilités en matière de communication, d'automatisation et de supervision. Cependant, leur nature distribuée, hétérogène et contrainte les rend également sensibles à plusieurs problèmes de sécurité. La diversité des attaques possibles, allant du déni de service aux comportements discrets comme le blackhole ou le selective forwarding, montre l'importance de disposer de mécanismes de détection adaptés.

Dans ce contexte, les systèmes IDS fondés sur l'intelligence artificielle constituent une approche pertinente pour automatiser la détection et améliorer la réactivité face aux menaces. L'utilisation de Cooja comme environnement de simulation offre un cadre expérimental flexible et reproductible, adapté à la validation de la solution proposée. Le chapitre suivant sera consacré à l'analyse du besoin et à l'étude de l'évolution du projet vers une architecture de détection en mode live.

Chapitre 2 : Analyse du besoin et étude de l'existant

2.1 Introduction

Après avoir présenté le cadre théorique général lié à l'Internet des Objets, aux enjeux de sécurité et aux systèmes de détection d'intrusion, il convient à présent d'analyser le besoin auquel répond ce projet. Ce chapitre vise à situer le travail dans son contexte fonctionnel, à mettre en évidence les limites des approches initiales envisagées, puis à justifier l'évolution progressive de la solution vers une architecture plus adaptée à la surveillance d'un environnement IoT dynamique.

L'objectif de cette analyse n'est pas uniquement de décrire les différentes étapes du projet, mais surtout de montrer comment la problématique de la protection d'un réseau IoT a conduit à la recherche d'une solution de détection plus cohérente, capable d'exploiter les données issues de la simulation, de traiter les événements observés et de restituer des résultats de manière claire. Cette réflexion permet ainsi de faire apparaître les besoins fonctionnels du système, les exigences liées à sa supervision et les choix ayant orienté l'évolution vers un mode de fonctionnement live.

2.2 Contexte et besoin du projet

Le présent projet s'inscrit dans le domaine de la sécurité des réseaux IoT, avec un intérêt particulier pour la protection du réseau contre différentes attaques à l'aide d'un système de détection d'intrusion (IDS)**[19], [20]**. Dans un tel contexte, il ne s'agit pas uniquement d'identifier des comportements anormaux, mais également de proposer une solution capable de surveiller l'activité du réseau, d'exploiter les données disponibles et de fournir des résultats compréhensibles pour l'utilisateur ou l'administrateur du système.

Le besoin principal du projet réside donc dans la conception d'un système IDS capable de renforcer la sécurité d'un environnement IoT simulé. Une telle solution doit être en mesure d'exploiter les traces d'activité générées par l'environnement de simulation, d'en extraire des carac-

téristiques pertinentes, d'appliquer un mécanisme de classification et de produire une décision de détection exploitable. Ce besoin ne se limite pas à la simple prédiction, mais inclut également la supervision des alertes, la visualisation des résultats et l'amélioration de la visibilité sur l'état global du réseau.

Dans cette perspective, l'environnement Cooja constitue un cadre particulièrement adapté, car il permet de reproduire des scénarios de communication et d'attaque dans un contexte contrôlé. Le recours à la simulation offre la possibilité d'observer le comportement du réseau, de produire des logs exploitables et d'évaluer le fonctionnement du système de détection sans dépendre directement d'un déploiement matériel réel.

Chaîne de traitement du système IDS proposé

Figure 2.1 – Chaîne de traitement globale : de la simulation Cooja à la supervision des alertes

2.3 Étude de l'approche initiale

Au début du projet, la détection des attaques reposait sur une approche hors ligne (offline) fondée sur un jeu de données statique. Le principe général consistait à partir d'un ensemble d'observations déjà collectées, à en extraire les variables retenues pour l'analyse, puis à appliquer un modèle de classification afin de produire une prédiction. Cette première démarche a constitué une étape utile dans la validation de la faisabilité du projet.

L'intérêt principal de cette approche initiale résidait dans la possibilité d'évaluer le comportement du modèle dans un cadre simple et maîtrisé. En travaillant sur un dataset statique, il devenait possible de vérifier si les variables sélectionnées présentaient effectivement un intérêt pour la détection, de mesurer la capacité du modèle à distinguer plusieurs types d'attaques et d'identifier les premiers points forts ou limites de la chaîne de traitement.

Cette approche a également joué un rôle structurant dans le projet. Elle a permis de clarifier les étapes essentielles du processus de détection — à savoir la disponibilité des données, l'extrac-

tion des variables, l'application du modèle et l'interprétation de la sortie — constituant ainsi une première démonstration du principe général de fonctionnement de l'IDS envisagé.

2.4 Limites de l'approche offline et évolution vers une architecture live

Malgré son utilité dans la phase initiale, l'approche hors ligne présentait plusieurs limites importantes qui ont motivé l'évolution de la solution. Le tableau comparatif ci-après synthétise les différences fondamentales entre les deux approches :

Tableau 2.1 – Comparaison entre l'approche offline et l'architecture live

Critère	Approche Offline	Architecture Live
Source de données	Dataset statique préparé à l'avance	Logs générés dynamiquement par Cooja
Moment d'analyse	Post-traitement, hors ligne	en continu, lors de la simulation
Réactivité	Aucune réactivité aux événements	Détection progressive et continue
Supervision	Résultats figés, non interactifs	Alertes affichées en temps quasi-réel
Adaptabilité	Limitée au dataset initial	Extensible à de nouveaux scénarios
Représentativité	Comportement réseau simplifié	Fidèle au comportement IoT dynamique

Premièrement, l'approche offline reposait sur des données figées, ne permettant pas de représenter fidèlement le comportement réel d'un environnement IoT en fonctionnement continu. Dans un réseau dynamique, les événements se produisent de manière évolutive et imprévisible, rendant insuffisante une analyse purement statique.

Deuxièmement, cette approche ne facilitait pas l'intégration d'une interface de supervision dynamique. Les prédictions étaient produites à partir d'un dataset préparé à l'avance, sans lien direct avec un flux de données généré en temps d'exécution. Enfin, le mode hors ligne ne répondait pas à la nécessité d'observer les événements lorsqu'ils se produisent et de transmettre les résultats dans une logique de surveillance continue.

2.5 Passage vers une architecture live

Afin de dépasser les limites identifiées, le projet a progressivement évolué vers une architecture live fondée sur l'exploitation continue des logs générés par Cooja. Dans cette nouvelle approche, les données ne sont plus considérées comme un ensemble statique, mais comme un flux d'événements à observer, à interpréter et à traiter au fur et à mesure de leur production.

Les logs issus de Cooja deviennent ainsi la source principale d'information. À partir de ces traces, la solution extrait les variables nécessaires, applique le modèle de classification et transmet les résultats vers une interface de supervision. Ce passage à une architecture live répond à plusieurs objectifs : rapprocher la chaîne de détection du fonctionnement réel d'un IDS, relier plus directement la simulation du réseau aux décisions de détection, et assurer un lien cohérent entre les événements observés, les features calculées, les prédictions du modèle et les alertes affichées.

2.6 Besoins fonctionnels et non fonctionnels

Pour répondre aux objectifs du projet, la solution proposée doit satisfaire un ensemble de besoins fonctionnels et non fonctionnels. Ces besoins traduisent à la fois ce que le système doit réaliser et les qualités qu'il doit présenter pour être cohérent avec le contexte de supervision et de détection envisagé.

Tableau 2.2 – Besoins fonctionnels du système IDS

ID	Besoin Fonctionnel	Description
BF1	Lecture des logs Cooja	Le système doit être capable de lire et parser les fichiers de logs générés par le simulateur Cooja lors de l'exécution des scénarios IoT.
BF2	Extraction de features	À partir des logs bruts, le système doit extraire automatiquement les variables réseau pertinentes nécessaires à la classification.
BF3	Classification ML	Le système doit appliquer le modèle d'apprentissage automatique entraîné afin de produire une décision de détection pour chaque observation.
BF4	Génération d'alertes	En cas de détection d'une attaque, le système doit gé-

		nérer une alerte identifiant le type d'attaque détecté.
BF5	Interface de supervision	Le système doit disposer d'un tableau de bord affichant en temps quasi-réel les prédictions, les alertes et les indicateurs de l'état du réseau.

Tableau 2.2 – Besoins fonctionnels du système IDS

Tableau 2.3 – Besoins non fonctionnels du système IDS

ID	Besoin Non Fonctionnel	Description
BNF1	Réactivité	Le traitement des logs et la génération des prédictions doivent s'effectuer avec un délai minimal pour garantir une détection en continu.
BNF2	Modularité	L'architecture doit être suffisamment modulaire pour permettre l'ajout de nouveaux modèles, de nouvelles features ou de nouveaux scénarios d'attaque.
BNF3	Lisibilité des résultats	L'interface de supervision doit présenter les résultats de manière claire et compréhensible pour un administrateur réseau.
BNF4	Maintenabilité	Le code et les composants doivent être organisés de façon à faciliter la maintenance, la correction et l'évolution future du système.
BNF5	Cohérence technique	L'ensemble de la chaîne (simulation, extraction, détection, supervision) doit maintenir une cohérence technique garantissant la fiabilité des résultats.

2.7 Conclusion

À travers ce chapitre, il apparaît que le projet répond à un besoin clair : renforcer la sécurité d'un environnement IoT simulé à l'aide d'un système de détection capable d'exploiter les données du réseau et de produire des résultats de manière exploitable. L'étude de l'approche initiale a montré qu'une logique hors ligne pouvait constituer une première étape utile pour valider le principe de classification et l'intérêt des variables retenues.

Toutefois, comme le montre le tableau comparatif présenté en section 2.4, cette démarche ne suffisait pas à refléter le comportement dynamique d'un réseau IoT ni à répondre pleinement

aux exigences d'une supervision continue. L'évolution vers une architecture live s'est ainsi imposée comme une réponse cohérente aux limites identifiées, permettant de relier la simulation, l'extraction des features, la détection et la supervision dans une même chaîne de traitement.

Les besoins fonctionnels et non fonctionnels identifiés en section 2.6 serviront de référentiel pour valider les choix de conception effectués. Ce travail d'analyse prépare naturellement le chapitre suivant, consacré à la conception générale du système et aux choix architecturaux retenus pour son implémentation

Chapitre 3 :Fondements théoriques du Machine Learning pour la détection d'intrusion

3.1Introduction

Dans les environnements IoT modernes, les systèmes de détection d'intrusion doivent être capables d'analyser de grandes quantités de données réseau afin d'identifier les comportements anormaux et les activités malveillantes. Les approches traditionnelles basées sur des signatures prédéfinies présentent certaines limites face à l'évolution constante des attaques et à la diversité des comportements observés dans les réseaux IoT.

Dans ce contexte, les techniques d'apprentissage automatique (Machine Learning) constituent une solution particulièrement adaptée pour améliorer les capacités de détection. Ces techniques permettent d'apprendre automatiquement des modèles à partir des données observées et de distinguer les comportements normaux des comportements suspects.

Ce chapitre présente les fondements théoriques du Machine Learning utilisés dans ce travail. Il introduit d'abord les concepts essentiels de l'apprentissage supervisé, puis décrit les principaux modèles de classification étudiés, notamment Decision Tree, K-Nearest Neighbors (KNN), Support Vector Machine (SVM) et Random Forest. Enfin, les métriques d'évaluation utilisées pour comparer les performances de ces modèles sont présentées.

3.2 Concepts fondamentaux du Machine Learning

Avant de présenter les modèles de classification utilisés dans ce travail, il est nécessaire de rappeler certains concepts fondamentaux liés à l'apprentissage automatique. Ces notions permettent de mieux comprendre le fonctionnement des algorithmes étudiés ainsi que leur rôle dans la détection d'intrusion au sein des réseaux IoT.

3.2.1 Intelligence Artificielle

L'intelligence artificielle (IA) désigne un ensemble de techniques et de méthodes permettant à une machine de reproduire certaines capacités généralement associées à l'intelligence humaine, telles que l'apprentissage, le raisonnement, la prise de décision ou encore la résolution de problèmes. Son objectif principal est de concevoir des systèmes capables d'analyser leur environnement et d'adapter leur comportement en fonction des informations disponibles.

Au cours des dernières années, l'intelligence artificielle a connu un développement considérable grâce à l'augmentation de la puissance de calcul et à la disponibilité de grandes quantités de données. Elle est aujourd'hui utilisée dans de nombreux domaines tels que la vision par ordinateur, le traitement automatique du langage naturel, les systèmes de recommandation, la cybersécurité et les réseaux intelligents.

Dans le domaine de la sécurité informatique, l'intelligence artificielle permet notamment d'automatiser l'analyse des données et de détecter des comportements suspects difficiles à identifier à l'aide des méthodes traditionnelles. Cette capacité est particulièrement intéressante dans les environnements IoT, où le volume et la diversité des données rendent la surveillance manuelle pratiquement impossible.

3.2.2 Machine Learning

Le Machine Learning, ou apprentissage automatique, constitue une branche de l'intelligence artificielle qui permet aux systèmes informatiques d'apprendre à partir des données sans être explicitement programmés pour chaque situation. Au lieu de suivre un ensemble fixe de règles définies à l'avance, les algorithmes de Machine Learning construisent automatiquement un modèle capable d'identifier des relations ou des tendances présentes dans les données observées.

Le principe général repose sur l'utilisation d'un ensemble d'exemples permettant au modèle d'apprendre les caractéristiques associées à différentes situations. Une fois l'apprentissage terminé, le modèle peut être utilisé pour effectuer des prédictions sur de nouvelles données jamais observées auparavant.

Les techniques de Machine Learning sont particulièrement adaptées aux systèmes de détection d'intrusion, car elles permettent d'analyser de grandes quantités de trafic réseau et de reconnaître automatiquement des comportements associés à des attaques ou à des anomalies. Elles offrent ainsi une solution efficace pour améliorer la sécurité des réseaux IoT face à l'évolution constante des menaces.

3.2.2 Machine Learning

Le Machine Learning, ou apprentissage automatique, constitue une branche de l'intelligence artificielle qui permet aux systèmes informatiques d'apprendre à partir des données sans être explicitement programmés pour chaque situation. Au lieu de suivre un ensemble fixe de règles définies à l'avance, les algorithmes de Machine Learning construisent automatiquement un modèle capable d'identifier des relations ou des tendances présentes dans les données observées.

Le principe général repose sur l'utilisation d'un ensemble d'exemples permettant au modèle d'apprendre les caractéristiques associées à différentes situations. Une fois l'apprentissage terminé, le modèle peut être utilisé pour effectuer des prédictions sur de nouvelles données jamais observées auparavant.

Les techniques de Machine Learning sont particulièrement adaptées aux systèmes de détection d'intrusion, car elles permettent d'analyser de grandes quantités de trafic réseau et de reconnaître automatiquement des comportements associés à des attaques ou à des anomalies. Elles offrent ainsi une solution efficace pour améliorer la sécurité des réseaux IoT face à l'évolution constante des menaces.

3.2.3 Apprentissage supervisé

L'apprentissage supervisé est l'une des approches les plus utilisées en Machine Learning. Dans cette méthode, le modèle est entraîné à partir d'un ensemble de données étiquetées, appelé dataset d'apprentissage. Chaque exemple est composé d'un ensemble de caractéristiques (features) ainsi que d'une classe ou étiquette (label) représentant le résultat attendu.

L'objectif du modèle est d'apprendre la relation existant entre les caractéristiques d'entrée et les classes de sortie afin de pouvoir prédire correctement la classe associée à de nouvelles données. Le processus d'apprentissage consiste à ajuster progressivement le modèle pour minimiser les erreurs de prédiction observées sur les données d'entraînement.

Dans le cadre de ce projet, les caractéristiques extraites à partir des logs générés par Cooja constituent les données d'entrée du modèle, tandis que les différentes catégories de trafic (Normal, DoS, Blackhole, Replay, etc.) représentent les classes à prédire.

3.2.4 Classification

La classification est une tâche d'apprentissage supervisé consistant à attribuer une observation à une catégorie prédéfinie. Le modèle reçoit un ensemble de caractéristiques décrivant une situation donnée et doit déterminer à quelle classe cette situation appartient.

Dans les systèmes de détection d'intrusion, la classification joue un rôle essentiel puisqu'elle permet d'identifier automatiquement le type de comportement observé sur le réseau. Selon les caractéristiques calculées à partir du trafic, le modèle peut déterminer si l'activité correspond à un fonctionnement normal ou à une attaque spécifique.

Dans ce travail, le problème étudié correspond à une classification multiclasse, puisque le système doit distinguer plusieurs catégories de trafic, incluant le trafic normal ainsi que différentes attaques réseau. Les algorithmes présentés dans les sections suivantes seront donc évalués dans ce contexte afin d'identifier la solution la plus adaptée aux exigences du système proposé.

3.3 Arbre de Décision (Decision Tree)

3.3.1 Définition

L'arbre de décision (Decision Tree) est un algorithme d'apprentissage supervisé largement utilisé pour les tâches de classification et de régression. Son principe consiste à représenter le processus de prise de décision sous la forme d'une structure arborescente composée de nœuds de décision et de feuilles terminales.

Chaque nœud interne correspond à un test effectué sur une caractéristique des données, tandis que chaque branche représente le résultat possible de ce test. Les feuilles de l'arbre correspondent aux classes finales prédites par le modèle.

Grâce à sa structure intuitive, l'arbre de décision est considéré comme l'un des modèles les plus simples à comprendre et à interpréter parmi les algorithmes de Machine Learning.

3.3.2 Principe de fonctionnement

Le fonctionnement d'un arbre de décision repose sur une série de divisions successives des données. À chaque étape, l'algorithme sélectionne la caractéristique qui permet de mieux séparer les différentes classes présentes dans le jeu de données.

Le processus débute au niveau de la racine de l'arbre. Les données sont ensuite divisées en plusieurs sous-ensembles selon la valeur d'une caractéristique donnée. Cette opération est répétée récursivement jusqu'à l'obtention de groupes suffisamment homogènes ou jusqu'à ce qu'un critère d'arrêt soit atteint.

L'objectif est de construire une structure permettant de classifier correctement les nouvelles observations en suivant un chemin allant de la racine vers une feuille finale.

3.3.3 Exemple simplifié

Dans le contexte d'un système de détection d'intrusion, un arbre de décision peut utiliser certaines caractéristiques du trafic réseau pour identifier le type d'activité observée.

Par exemple :

- Si le taux de perte des paquets est très élevé, le trafic peut être classé comme une attaque Blackhole.
- Si le délai moyen de réponse est anormalement important, le trafic peut être associé à une attaque Delay.
- Sinon, le trafic peut être considéré comme normal.

Chaque décision est prise progressivement jusqu'à l'obtention de la classe finale.

3.3.4 Critères de division

Afin de déterminer la meilleure caractéristique à utiliser lors de chaque division, plusieurs critères peuvent être employés. Les plus courants sont :

- L'indice de Gini (Gini Impurity)
- L'entropie (Entropy)
- Le gain d'information (Information Gain)

Ces mesures permettent d'évaluer la qualité d'une séparation en mesurant le niveau de pureté des groupes obtenus après la division.

L'algorithme choisit généralement la caractéristique qui maximise la séparation entre les différentes classes.

3.3.5 Avantages

Les principaux avantages des arbres de décision sont :

- Simplicité de compréhension et d'interprétation ;
- Facilité de visualisation ;
- Peu de prétraitement des données ;
- Temps de prédiction généralement rapide ;
- Adaptation aux problèmes de classification multiclasse.

3.3.6 Limites

Malgré ses avantages, l'arbre de décision présente certaines limites :

- Sensibilité aux variations des données d'entraînement ;
- Risque important de surapprentissage (Overfitting) ;
- Généralisation parfois limitée sur de nouvelles données ;
- Performance inférieure à celle des méthodes d'ensemble telles que Random Forest.

Ces limitations ont conduit au développement d'approches plus robustes basées sur l'agrégation de plusieurs arbres de décision, comme l'algorithme Random Forest qui sera présenté dans la section suivante.

3.4 Random Forest

3.4.1 Définition

Le Random Forest est un algorithme d'apprentissage supervisé appartenant à la famille des méthodes d'ensemble (Ensemble Learning). Il repose sur la combinaison de plusieurs arbres de

décision afin d'améliorer les performances de classification et de réduire les risques de surapprentissage observés avec un arbre unique.

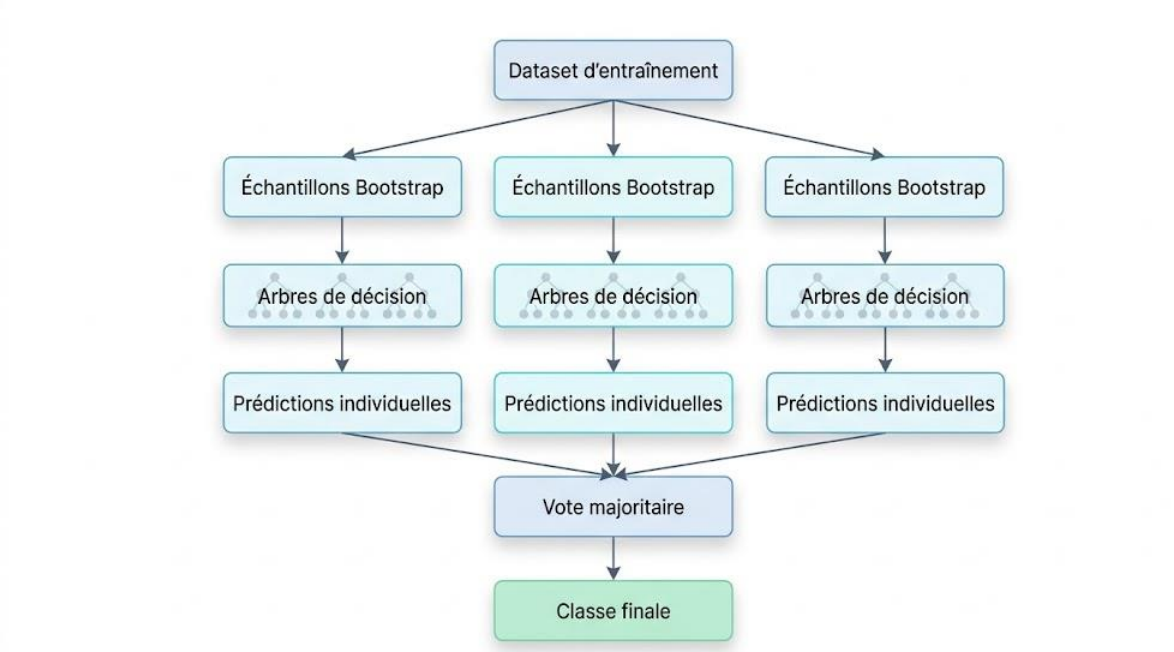
L'idée principale consiste à construire un grand nombre d'arbres de décision indépendants, puis à combiner leurs prédictions pour obtenir une décision finale plus robuste et plus fiable.

Grâce à cette approche collective, Random Forest est aujourd'hui considéré comme l'un des algorithmes de classification les plus performants pour les données tabulaires et les applications de cybersécurité.

3.4.2 Principe général

Contrairement à un arbre de décision classique qui repose sur une seule structure de décision, Random Forest construit plusieurs arbres différents à partir de sous-ensembles aléatoires des données d'apprentissage.

Chaque arbre produit sa propre prédiction de manière indépendante. Une fois toutes les prédictions obtenues, le modèle applique un mécanisme de vote majoritaire afin de déterminer la classe finale.



Le principe peut être résumé comme suit :

Figure3.1 : Principe général de l'algorithme Random Forest

3.4.3 Bootstrap Sampling

La première étape du Random Forest consiste à générer plusieurs sous-ensembles de données à partir du dataset d'origine.

Cette technique est appelée **Bootstrap Sampling**.

Chaque arbre est entraîné sur un échantillon obtenu par tirage aléatoire avec remise. Cela signifie qu'une même observation peut être sélectionnée plusieurs fois, tandis que d'autres peuvent ne pas être choisies.

Cette diversité des données permet de produire des arbres différents et de limiter leur corrélation.

Exemple:

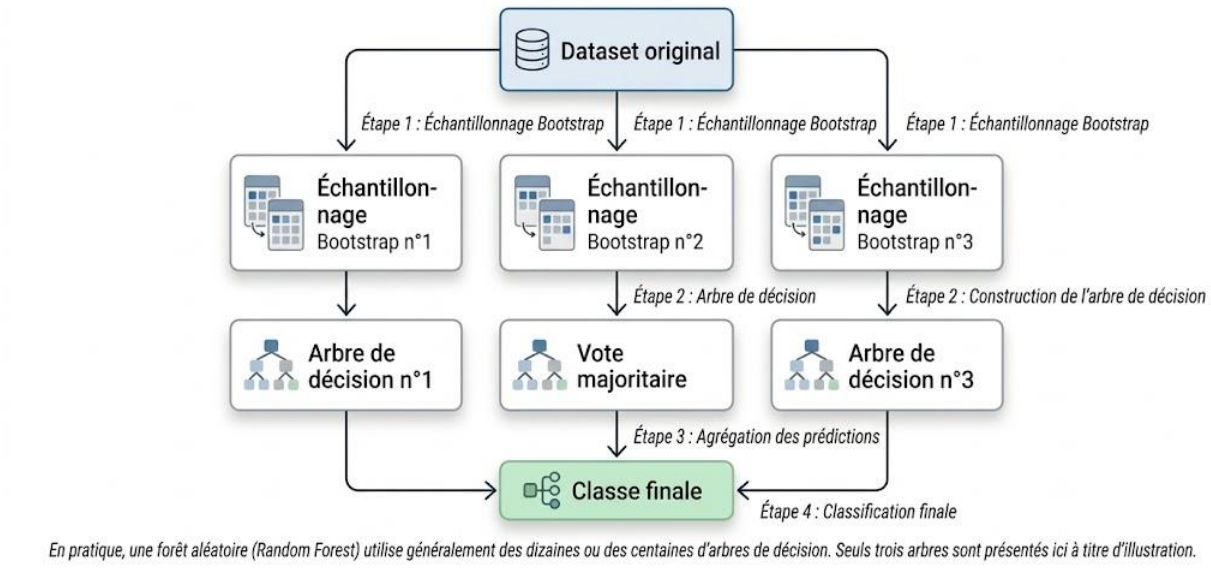


Figure 3.2 : Principe de fonctionnement de l'algorithme Forêt Aléatoire (Random Forest) basé sur l'Échantillonnage Bootstrap, la construction de multiples arbres de décision et le vote majoritaire.

Chaque arbre apprend donc sur des données légèrement différentes.

3.4.4 Sélection aléatoire des caractéristiques

En plus de l'échantillonnage des données, Random Forest introduit un second niveau d'aléa.

Lors de chaque division d'un arbre, l'algorithme ne considère qu'un sous-ensemble aléatoire des caractéristiques disponibles.

Par exemple, si le dataset contient 10 features :

matched_responses

avg_response_delay_ms

p95_delay

recv_ratio

loss_ratio

sent_count

rate_sent

recv_count

rate_recv

loss

l'arbre peut n'utiliser que 3 ou 4 caractéristiques parmi celles-ci pour rechercher la meilleure séparation.

Cette stratégie permet :

- d'augmenter la diversité des arbres ;
- de réduire le risque de surapprentissage ;
- d'améliorer la capacité de généralisation.

3.4.5 Vote majoritaire

Après l'entraînement, chaque arbre effectue sa propre prédiction.

Supposons que nous ayons cinq arbres

Arbre	Prédiction
Tree 1	DoS
Tree 2	DoS
Tree 3	Normal
Tree 4	DoS
Tree 5	Blackhole

Le résultat final est obtenu par vote majoritaire.Ici :

DoS = 3 votes

Normal = 1 vote

Blackhole = 1 vote

La prédiction finale sera donc : **Dos**

Mathématiquement :

où :

Ti(x) représente la prédiction du i^e arbre ;

- y^ représente la classe finale ;
- mode désigne la valeur apparaissant le plus fréquemment.
- mode désigne la valeur apparaissant le plus fréquemment.

3.4.6 Algorithme de Random Forest

Le fonctionnement global du modèle peut être résumé par les étapes suivantes :

Étape 1

Collecte du dataset d'apprentissage.

Étape 2

Création de plusieurs échantillons Bootstrap.

Étape 3

Construction d'un arbre de décision pour chaque échantillon.

Étape 4

Sélection aléatoire d'un sous-ensemble de caractéristiques à chaque nœud.

Étape 5

Apprentissage indépendant de chaque arbre.

Étape 6

Prédiction individuelle de tous les arbres.

Étape 7

Application du vote majoritaire.

Étape 8

Production de la classe finale.

3.4.7 Avantages du Random Forest

Random Forest présente plusieurs avantages importants :

- Très bonne précision de classification ;
- Réduction du surapprentissage ;
- Robustesse face au bruit ;
- Gestion naturelle des problèmes multiclasses ;
- Bonne capacité de généralisation ;
- Calcul rapide lors de la phase de prédiction ;
- Possibilité d'évaluer l'importance des variables.

Ces caractéristiques expliquent son utilisation fréquente dans les systèmes IDS basés sur le Machine Learning.

3.4.8 Limites du Random Forest

Malgré ses performances, Random Forest présente certaines limites :

- Temps d'entraînement plus élevé qu'un arbre unique ;
- Consommation mémoire plus importante ;
- Interprétation plus complexe qu'un simple Decision Tree ;
- Difficulté à expliquer individuellement chaque décision.

Cependant, ces limites restent généralement acceptables au regard du gain de performance obtenu.

3.4.9 Pourquoi Random Forest pour ce projet ?

Dans le cadre de ce travail, plusieurs modèles de classification ont été étudiés. Le choix du Random Forest s'explique par plusieurs raisons :

- capacité à traiter efficacement des données tabulaires issues des logs réseau ;
- bonne gestion des problèmes multiclasses (Normal, DoS, Blackhole, Replay, etc.) ;
- robustesse face au bruit présent dans les données simulées ;
- temps d'inférence compatible avec un fonctionnement en **quasi temps réel** ;
- performances reconnues dans les travaux de détection d'intrusion.

Pour ces raisons, Random Forest a été retenu comme modèle principal du système proposé.

3.5 K-Nearest Neighbors (KNN)

3.5.1 Définition

L'algorithme K-Nearest Neighbors (KNN) est une méthode d'apprentissage supervisé utilisée pour les tâches de classification et de régression. Contrairement à de nombreux modèles d'apprentissage, KNN ne construit pas explicitement un modèle lors de la phase d'entraînement. Il conserve simplement les données d'apprentissage et réalise les prédictions en recherchant les observations les plus proches d'un nouvel exemple.

Le principe fondamental repose sur l'hypothèse selon laquelle des observations similaires appartiennent généralement à la même classe.

3.5.2 Principe de fonctionnement

Lorsqu'une nouvelle observation doit être classifiée, l'algorithme calcule sa distance par rapport à toutes les observations du dataset d'apprentissage.

Il sélectionne ensuite les K voisins les plus proches puis attribue à l'observation la classe majoritaire parmi ces voisins.

Par exemple, si K = 5 :

- 3 voisins appartiennent à la classe DoS
- 1 voisin appartient à la classe Blackhole
- 1 voisin appartient à la classe Normal

La prédiction finale sera :

Classe = DoS

3.5.3 Distance Euclidienne

Pour évaluer la similarité entre les observations, la mesure la plus couramment utilisée est la distance euclidienne. Elle se calcule à l'aide de la formule suivante :

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Où :

- xi: représente la nouvelle observation à classifier.
- yi : représente une observation existante dans le dataset (un voisin).
- n : représente le nombre total de caractéristiques (features) extraites du trafic réseau.
- xi et yi: représentent les valeurs de la i-ème caractéristique pour les observations xi et yi.

3.5.4 Avantages

- Simple à comprendre ;
- Facile à implémenter ;
- Aucun apprentissage complexe ;

- Bon comportement sur des datasets de petite taille.

3.5.5 Limites

- Temps d'inférence élevé ;
- Sensible aux données bruitées ;
- Consommation mémoire importante ;
- Peu adapté aux systèmes de détection quasi temps réel.

3.6 Support Vector Machine (SVM)

3.6.1 Définition

Le Support Vector Machine (SVM), ou Machine à Vecteurs de Support, est un algorithme d'apprentissage supervisé robuste utilisé pour les tâches de classification et de régression. Son objectif principal est de construire une frontière de séparation optimale (appelée hyperplan) entre différentes classes afin de maximiser la marge de distinction. L'idée fondamentale consiste à projeter les données dans un espace de dimension supérieure (si nécessaire) pour trouver un hyperplan séparateur, tout en maximisant la distance entre les observations les plus proches des différentes catégories.

3.6.2 Principe de fonctionnement

Le fonctionnement du SVM repose sur la recherche de l'hyperplan qui maximise la marge, c'est-à-dire la distance entre la frontière de décision et les points les plus proches de chaque classe. Les observations situées sur les limites de cette marge sont appelées les **Vecteurs de support (Support Vectors)**. Ce sont ces points critiques qui déterminent la position et l'orientation de l'hyperplan ; la suppression des autres points du dataset ne modifierait en rien la frontière trouvée.

Mathématiquement, l'hyperplan de séparation dans un espace linéaire est défini par l'équation suivante :

$$w \cdot x + b = 0$$

w représente le vecteur de poids (normal à l'hyperplan, déterminant son orientation).

- x représente le vecteur de caractéristiques de l'observation réseau (ex: loss_ratio, sent_count).
- b représente le biais (bias, déterminant la distance de l'hyperplan par rapport à l'origine).

Dans le contexte d'un système de détection d'intrusion (IDS) pour l'IoT, le SVM cherche à tracer cet hyperplan pour séparer le trafic **Normal** du trafic malveillant (comme les attaques **DoS** ou **Blackhole**) en se basant sur les signatures statistiques extraites.

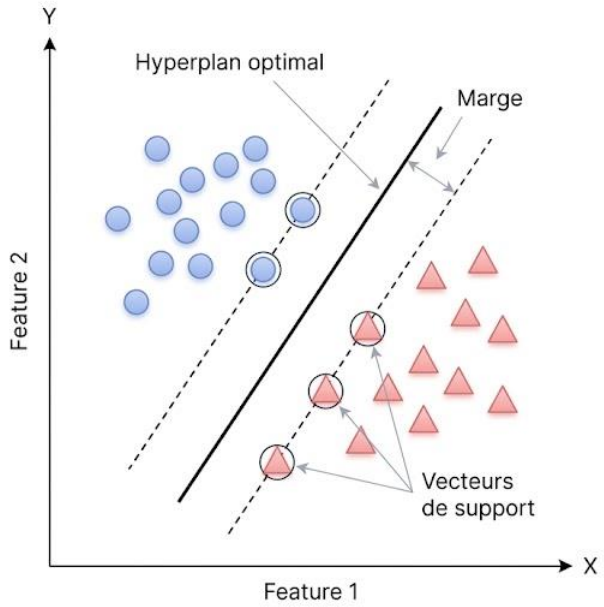


Figure 3.3 : Principe de séparation linéaire et vecteurs de support dans un modèle SVM

3.6.3 Avantages

- **Bonne capacité de généralisation :** La maximisation de la marge permet de réduire efficacement le risque de surapprentissage (Overfitting).
- **Efficacité en grande dimension :** Performant même lorsque le nombre de caractéristiques (features) extraites du réseau est élevé.
- **Flexibilité grâce aux Kernels :** Capacité à traiter des données non linéairement séparables en utilisant des fonctions de noyau (*Kernel tricks*) pour projeter les données dans un espace à plus haute dimension où elles deviennent linéairement séparables.

3.6.4 Limites

- **Paramétrage complexe :** Nécessite un choix judicieux de la fonction de noyau (linéaire, RBF, polynomial) et un ajustement rigoureux des hyperparamètres (comme le paramètre de régularisation C).
- **Temps d'entraînement élevé :** Le coût de calcul devient prohibitif et extrêmement lourd sur de grands ensembles de données (Datasets volumineux).
- **Complexité d'interprétation :** Contrairement aux arbres de décision, le modèle SVM fonctionne mathématiquement comme une boîte noire, rendant l'explication directe des décisions de classification difficile pour l'administrateur réseau.
- **Limite face au trafic multiclasse en temps réel :** Conçu initialement pour la classification binaire (deux classes). Pour étendre le SVM au cas multiclasse de notre IDS (Normal, DoS, Blackhole, Delay), il est nécessaire de combiner plusieurs classificateurs (approches *One-vs-One* ou *One-vs-Rest*), ce qui augmente considérablement le temps d'inférence et le rend moins adapté qu'un modèle *Random Forest* pour une détection quasi temps réel dans les environnements IoT contraints en ressources.

3.7 Métriques d'évaluation

Pour évaluer les performances d'un modèle d'apprentissage automatique, il est primordial de ne pas se limiter à une seule métrique. Dans le contexte de la détection d'intrusion (IDS), le trafic réseau est souvent déséquilibré (le trafic normal étant majoritaire par rapport aux attaques). Par conséquent, s'appuyer uniquement sur l'exactitude (Accuracy) peut s'avérer trompeur.

L'évaluation repose sur les éléments fondamentaux de la matrice de confusion :

- **Vrais Positifs (TP - True Positives) :** Attaques correctement détectées.
- **Vrais Négatifs (TN - True Negatives) :** Trafic normal correctement classifié.
- **Faux Positifs (FP - False Positives) :** Trafic normal faussement détecté comme une attaque (Fausse alerte).
- **Faux Négatifs (FN - False Negatives) :** Attaque réelle non détectée par le système.

Sur la base de ces éléments, les métriques suivantes ont été retenues pour l'évaluation de nos modèles :

3.7.1 Exactitude (Accuracy)

L'accuracy représente la proportion totale des prédictions correctes (positives et négatives) parmi l'ensemble des prédictions réalisées par le modèle. Bien qu'elle donne un aperçu général de la performance, elle reste insuffisante en cas de classes déséquilibrées.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

3.7.2 Précision (Precision)

La précision mesure la proportion des prédictions positives qui sont effectivement correctes. Dans un IDS, une haute précision signifie que lorsque le système déclenche une alerte d'intrusion (ex: attaque DoS ou Blackhole), il est hautement probable qu'il s'agisse d'une véritable attaque. Elle permet de quantifier les fausses alertes (FP).

$$\text{Precision} = \frac{TP}{TP + FP}$$

3.7.3 Rappel (Recall / Sensibilité)

Le Recall mesure la capacité du modèle à détecter toutes les instances réellement positives. Dans le contexte de la sécurité IoT, cette métrique est critique : un Recall élevé garantit qu'un minimum d'attaques a réussi à contourner le système sans être détecté (minimisation des FN). Il vaut souvent mieux avoir quelques fausses alertes qu'une véritable attaque qui passe inaperçue.

$$\text{Recall} = \frac{TP}{TP + FN}$$

3.7.4 Score F1 (F1-Score)

Le F1-Score est la moyenne harmonique de la précision et du rappel. Il permet de trouver un compromis optimal entre ces deux métriques, en particulier lorsque le dataset présente une distribution inégale des classes (par exemple, beaucoup de paquets normaux et peu de paquets modifiés par une attaque Delay). Un F1-Score élevé indique un modèle à la fois précis et exhaustif.

$$F1\text{-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

3.7.5 Temps d'inférence

Le temps d'inférence correspond au temps nécessaire au modèle pour analyser de nouvelles données entrantes et produire une prédiction (classification). Dans le cadre d'un IDS appliqué à l'Internet des Objets (IoT) et fonctionnant en quasi temps réel, ce critère est d'une importance capitale. Les décisions doivent être produites en quelques millisecondes afin d'identifier et de bloquer les nœuds malveillants sans introduire de délai excessif dans le réseau, justifiant ainsi le choix d'algorithmes rapides lors de la phase de prédiction (comme Random Forest) par rapport à des modèles plus lents (comme SVM ou KNN).

3.8 Étude comparative et sélection du modèle

Afin de sélectionner l'algorithme le plus adapté à notre système de détection d'intrusion (IDS), une étude comparative a été réalisée entre plusieurs modèles de Machine Learning couramment utilisés pour les tâches de classification. Les algorithmes étudiés sont : Decision Tree (DT), K-Nearest Neighbors (KNN), Support Vector Machine (SVM) et Random Forest (RF).

L'évaluation a été effectuée à l'aide des métriques présentées précédemment, à savoir l'Accuracy, la Precision, le Recall, le F1-Score ainsi que le temps d'inférence. Ce dernier constitue un critère particulièrement important dans le contexte d'un système IDS destiné à fonctionner en quasi temps réel dans un environnement IoT.

Tableau 3.1 : Performances comparatives des algorithmes de Machine Learning

Modèle	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Temps d'inférence (ms/échantillon)
K-Nearest Neighbors (KNN)	86.93	86.86	86.93	86.39	0.23
Random Forest (RF)	81.24	80.19	81.24	80.09	0.02
Decision Tree (DT)	81.11	79.99	81.11	79.91	0
Support Vector Machine (SVM)	**63.97**	**60.24**	**63.97**	**55**	**4.24**

3.8.1 Analyse des résultats

Les résultats obtenus montrent que l'algorithme KNN présente les meilleures performances globales en termes d'Accuracy, de Precision, de Recall et de F1-Score. Avec une accuracy de 86.93 %, il constitue le modèle le plus performant du point de vue de la qualité de classification.

Cependant, le mécanisme de prédiction du KNN repose sur la comparaison de chaque nouvelle observation avec l'ensemble des données d'apprentissage. Cette caractéristique entraîne une augmentation progressive du temps de calcul lorsque la taille du dataset devient importante. Dans un contexte de surveillance continue du trafic réseau, cette limitation peut affecter la réactivité du système.

Le modèle SVM présente les performances les plus faibles parmi les algorithmes évalués. Son accuracy de 63.97 % ainsi que son temps d'inférence élevé montrent qu'il est moins adapté au problème étudié. Cette situation s'explique notamment par la nature multiclasse du dataset et par la complexité des comportements réseau générés par les différentes attaques.

Le Decision Tree obtient des résultats proches de ceux du Random Forest tout en offrant un temps d'inférence extrêmement faible. Toutefois, étant basé sur un arbre unique, il demeure généralement plus sensible aux variations des données et moins robuste que les méthodes d'ensemble.

3.8.2 Justification du choix du Random Forest

Bien que le modèle KNN obtienne les meilleures performances en termes de précision globale, le choix final de notre système s'est porté sur le Random Forest pour plusieurs raisons techniques et opérationnelles.

Rapidité d'inférence : Le Random Forest présente un temps d'inférence très faible (0.0214 ms par échantillon), compatible avec les exigences d'un système IDS fonctionnant en quasi temps réel.

Robustesse et stabilité : Grâce à son principe d'ensemble basé sur plusieurs arbres de décision et sur le vote majoritaire, le Random Forest est moins sensible aux variations des données et au bruit présent dans les communications réseau.

Passage à l'échelle (Scalability) : Contrairement au KNN, dont le temps de prédiction augmente avec la taille du dataset, le Random Forest conserve des performances stables même lorsque le volume des données devient important.

Interprétabilité : Le Random Forest permet d'évaluer l'importance des variables (Feature Importance), ce qui facilite l'analyse du comportement du modèle et l'identification des caractéristiques les plus influentes dans la détection des attaques.

En conséquence, le Random Forest représente le meilleur compromis entre précision, robustesse, interprétabilité et rapidité d'exécution. Ces caractéristiques le rendent particulièrement adapté à l'architecture IDS proposée dans ce projet et justifient son intégration comme modèle principal de détection.

3.9 Conclusion

Dans ce chapitre, les principaux fondements théoriques du Machine Learning appliqué à la détection d'intrusion ont été présentés. Après avoir introduit les concepts de l'apprentissage supervisé, plusieurs algorithmes de classification ont été étudiés, notamment le Decision Tree, le K-Nearest Neighbors, le Support Vector Machine et le Random Forest.

Une attention particulière a été accordée au Random Forest, qui constitue le modèle principal retenu dans le cadre de ce projet. Son fonctionnement, basé sur la combinaison de plusieurs arbres de décision et sur le mécanisme de vote majoritaire, lui permet d'améliorer la robustesse des prédictions tout en limitant les risques de surapprentissage.

Les différentes métriques d'évaluation utilisées pour mesurer les performances des modèles ont également été présentées. Cette étude a montré qu'une évaluation pertinente d'un système IDS ne peut pas se limiter à l'Accuracy, mais doit également prendre en compte la Precision, le Recall, le F1-Score ainsi que le temps d'inférence, particulièrement important dans les environnements IoT fonctionnant en quasi temps réel.

Enfin, l'étude comparative réalisée entre les différents algorithmes a permis de mettre en évidence les avantages et les limites de chaque approche. Bien que le modèle KNN ait obtenu les meilleures performances de classification, le Random Forest a été retenu en raison de son excellent compromis entre précision, robustesse, interprétabilité et rapidité d'exécution. Ces caractéristiques le rendent particulièrement adapté à l'architecture IDS proposée dans ce travail.

Les éléments théoriques présentés dans ce chapitre constituent ainsi la base scientifique nécessaire à la compréhension des choix techniques effectués dans la suite du projet. Le chapitre suivant sera consacré à la conception générale du système proposé et à la description détaillée de son architecture.

Chapitre 4 : Conception générale du système

4.1 Introduction

Après avoir analysé le besoin auquel répond ce projet ainsi que l'évolution progressive de la solution vers un mode de fonctionnement live, il convient à présent de présenter la conception générale du système proposé. Ce chapitre a pour objectif de décrire l'architecture retenue, les principaux composants qui la constituent, ainsi que les relations fonctionnelles qui assurent le déroulement de la chaîne de détection.

La conception présentée repose sur une approche cohérente reliant l'environnement de simulation IoT, l'exploitation des logs générés, l'extraction des caractéristiques pertinentes, l'application du modèle de classification et la transmission des résultats vers une interface de supervision. Il ne s'agit pas uniquement de décrire une succession d'outils techniques, mais de montrer comment l'ensemble des composants coopèrent pour construire un système de détection d'intrusion adapté à un environnement IoT simulé**[21]**.

4.2 Vue d'ensemble de l'architecture proposée

L'architecture proposée repose sur une chaîne de traitement orientée live, conçue pour assurer la détection d'attaques à partir des logs générés par l'environnement de simulation Cooja. Elle relie de manière cohérente la production des événements réseau, leur interprétation, leur transformation en variables exploitables, l'application du modèle de classification et la restitution des résultats à travers une interface de supervision.

De manière générale, le système s'articule autour de cinq blocs complémentaires. L'environnement Cooja joue le rôle de source de simulation en reproduisant le comportement d'un réseau IoT. Ces événements sont enregistrés dans le fichier cooja_live.log, qui constitue la matière première de la détection. Le backend app.py assure ensuite la lecture des logs, l'extraction des features, l'application du modèle de classification et la génération des alertes. Les résultats sont finalement transmis en continu vers le dashboard via WebSocket afin d'assurer une supervision continue.

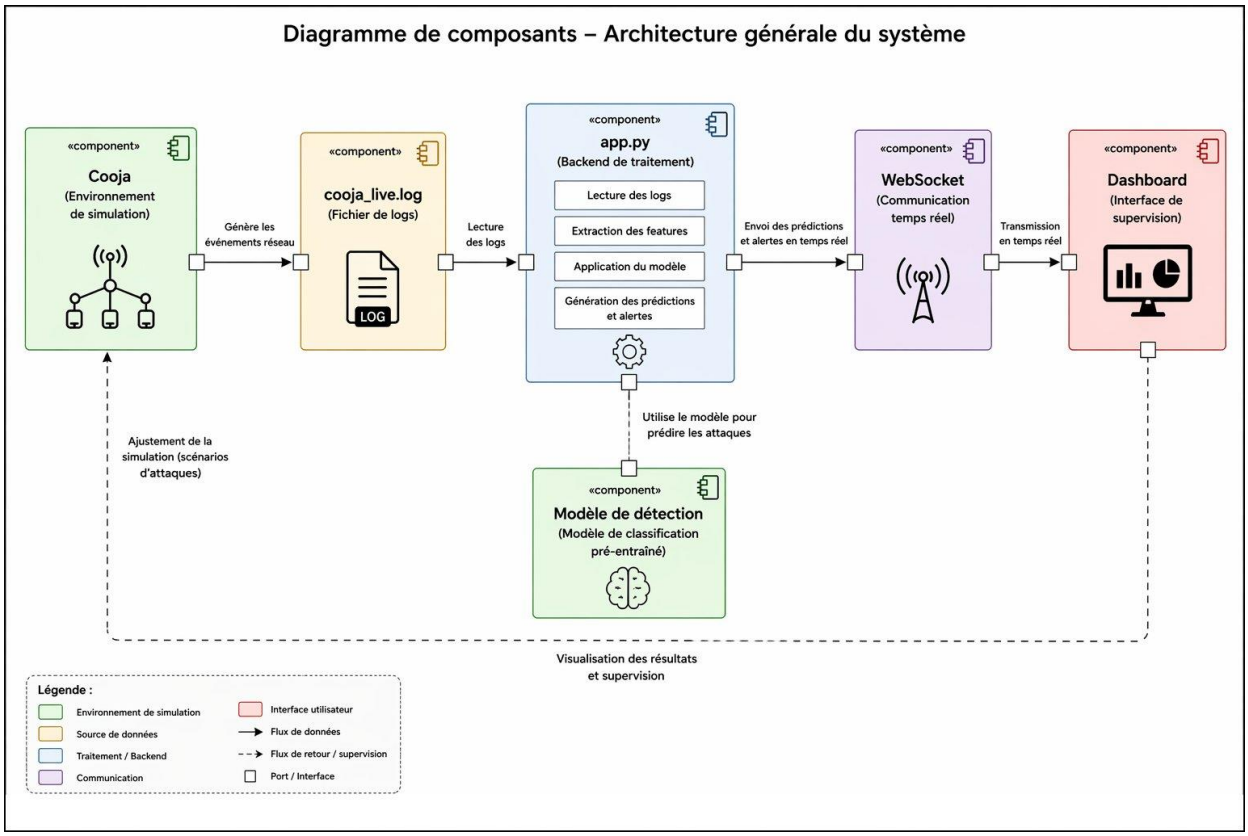


Figure 4.1 – Diagramme de composants : Architecture générale du système proposé

Comme le montre la Figure 3.1, la chaîne de traitement suit un flux unidirectionnel depuis la simulation jusqu'à la supervision, avec une boucle de retour permettant l'ajustement des scénarios d'attaque. L'architecture modulaire adoptée permet de délimiter clairement les responsabilités de chaque composant, ce qui facilite la maintenance et l'évolution du système.

Tableau 4.1 – Synthèse des composants de l'architecture

Composant	Rôle dans la chaîne	Technologie	Interface
Cooja Simulator	Simulation du réseau IoT et génération des événements réseau	Contiki-NG / Java	Fichier de logs
cooja_live.log	Stockage brut des événements réseau générés dy-	Fichier texte dynamique	Lecture par app.py

	namiquement		
app.py (Backend)	Lecture des logs, extraction des features, application du modèle ML, génération des alertes	Python 3 (scikit-learn / ONNX)	WebSocket
Modèle de détection	Classification des observations : trafic normal ou type d'attaque	Modèle ML pré-entraîné (ONNX)	Appelé par app.py
WebSocket Server	Communication temps réel entre backend et dashboard	asyncio / websockets	Port dédié
Dashboard	Supervision et visualisation des alertes et prédictions	HTML / JS / Chart.js	Navigateur web

4.3 Description des composants du système

Le système repose sur six composants complémentaires, chacun assurant une fonction précise dans la chaîne de détection. La compréhension de ces composants est essentielle pour distinguer les rôles respectifs de la simulation, du traitement, de la communication et de la supervision.

4.3.1 Environnement de simulation — Cooja

Cooja est le simulateur réseau intégré à Contiki-NG, conçu pour reproduire le fonctionnement de réseaux de capteurs IoT dans un cadre contrôlé. Il permet de déployer virtuellement plusieurs nœuds communicants et d'y injecter différents scénarios d'attaques (flooding, blackhole, sinkhole, etc.)[22]. Dans ce projet, Cooja joue le rôle de source expérimentale en générant les événements réseau observés et en les écrivant dans le fichier cooja_live.log. Son principal avantage réside dans la reproductibilité des expériences et l'absence de dépendance à un déploiement matériel réel.

4.3.2 Source des données — cooja_live.log

Le fichier cooja_live.log constitue la base d'entrée de tout le traitement. Il est généré dynamiquement par Cooja et contient les événements réseau sous forme de lignes textuelles horodatées, incluant des informations sur les nœuds source et destination, le type de paquet, la qualité du signal (RSSI), l'indice de qualité du lien (LQI) et la taille des paquets échangés. Le backend surveille ce fichier en continu et lit les nouvelles lignes au fur et à mesure de leur apparition, assurant ainsi le caractère live de la détection.

4.3.3 Module de traitement et de prédiction — app.py

Le module app.py représente le cœur fonctionnel du système. Il assure successivement la lecture des nouvelles lignes du fichier de logs, leur parsing via un module dédié, l'extraction des features pertinentes à partir d'une fenêtre glissante d'observations, l'application du modèle de classification pré-entraîné et la génération des prédictions associées à un niveau d'alerte**[23], [24]**. Ce module intègre également la logique de gestion des fenêtres temporelles, permettant de regrouper les événements réseau avant de produire une décision de détection.

4.3.4 Module de communication — WebSocket

La communication entre le backend et l'interface de supervision est assurée en temps réel via un serveur WebSocket. Ce choix technique permet d'établir une connexion persistante et bidirectionnelle entre app.py et le dashboard, garantissant une transmission fluide des prédictions et des alertes sans recourir à un mécanisme de polling périodique. Chaque nouvelle prédiction est immédiatement diffusée à tous les clients connectés, assurant ainsi la réactivité de la supervision.

4.3.5 Interface de supervision — Dashboard

Le dashboard constitue le point de contact entre le système de détection et l'administrateur réseau. Il reçoit les résultats transmis par le WebSocket et les affiche sous forme de graphiques, d'indicateurs et d'alertes visuelles. L'interface présente notamment l'historique des prédictions,

la distribution des types d'attaques détectées, le niveau d'alerte courant et des indicateurs de performance du réseau simulé. La conception du dashboard en composante distincte du back-end garantit une séparation claire des responsabilités et simplifie la maintenance de l'ensemble.

4.4 Cas d'utilisation du système

Avant de détailler le fonctionnement dynamique du système, il convient d'identifier les principaux cas d'utilisation structurant les interactions entre les acteurs et les composants. Le tableau suivant synthétise ces cas d'utilisation en distinguant les acteurs impliqués, les actions réalisées et leur description fonctionnelle.

Tableau 4.2 – Cas d'utilisation du système IDS

Acteur	Cas d'utilisation	Description
Cooja (système)	Générer des événements réseau	Simuler le trafic IoT normal et les scénarios d'attaques
Backend (app.py)	Lire et parser les logs	Surveiller en continu cooja_live.log et extraire les lignes valides
Backend (app.py)	Extraire les features	Calculer les variables réseau pertinentes à partir des entrées parsées
Backend (app.py)	Classer le trafic	Appliquer le modèle ML et produire une prédiction (normal / attaque)
Backend (app.py)	Générer des alertes	Créer une alerte avec niveau de sévérité en cas d'attaque détectée
Administrateur	Superviser le réseau	Consulter le dashboard pour visualiser alertes, prédictions et indicateurs
Administrateur	Interpréter les résultats	Analyser les patterns d'attaques détectées et l'état global du réseau

4.5 Fonctionnement global du système

Le fonctionnement global du système repose sur une chaîne continue de traitement qui commence par la génération d'événements réseau dans l'environnement de simulation et se termine par l'affichage des résultats dans l'interface de supervision. Cette organisation reflète la volonté de construire une solution cohérente, dans laquelle chaque composant intervient à un moment précis du processus.

4.5.1 Diagramme de séquence du fonctionnement global

Le fonctionnement dynamique du système peut être représenté par le diagramme de séquence ci-après, qui illustre l'enchaînement temporel des échanges entre les différents composants.

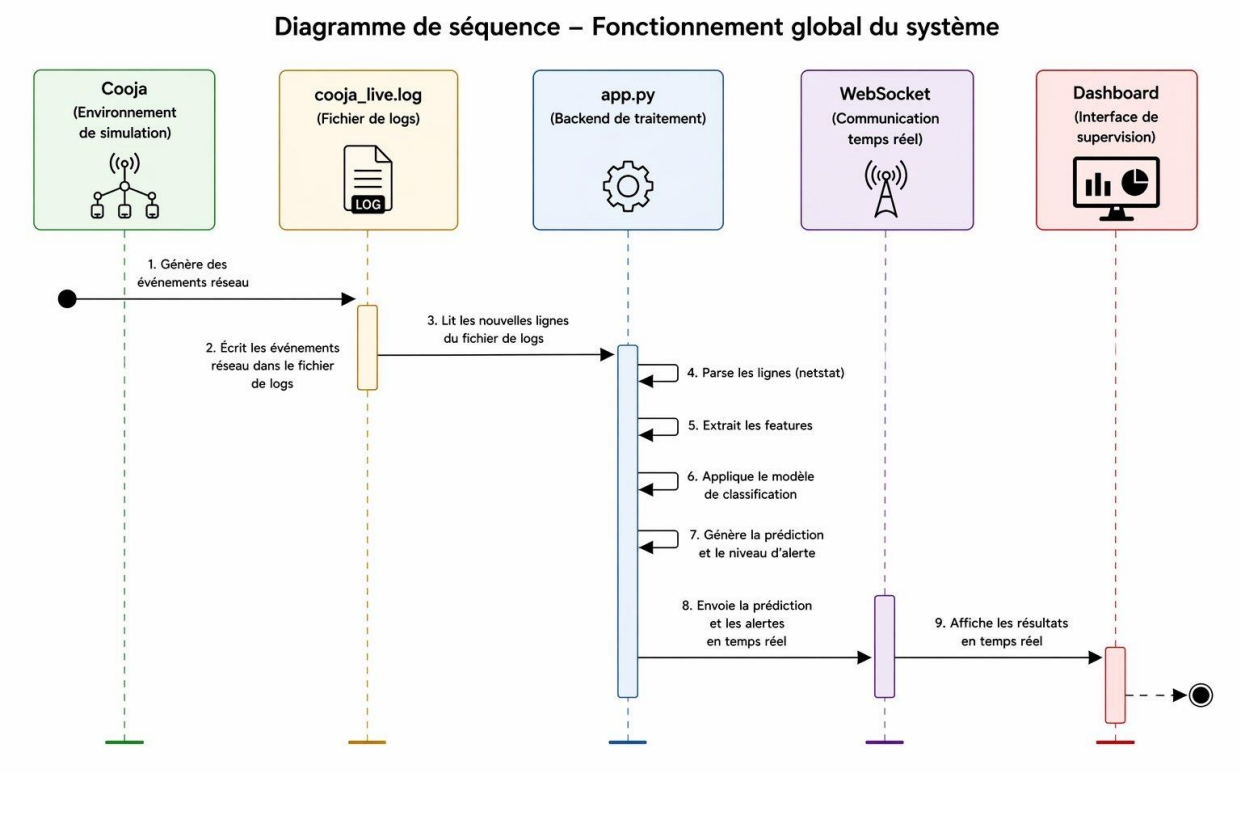


Figure 4.2 – Diagramme de séquence : Fonctionnement global du système

Ce diagramme illustre les neuf étapes séquentielles du traitement. Cooja génère les événements réseau (étape 1) et les écrit dans le fichier de logs (étape 2). Le backend lit les nouvelles lignes (étape 3), les parse (étape 4), extrait les features (étape 5), applique le modèle de classification (étape 6) et génère la prédiction avec son niveau d'alerte (étape 7). Le résultat est transmis via WebSocket (étape 8) et affiché en temps réel sur le dashboard (étape 9). La représentation séquentielle met en évidence le rôle central d'app.py, qui assure à la fois l'analyse des logs, la prédiction et la génération des alertes.

4.5.2 Diagramme d'activités du processus de détection

Le déroulement logique du traitement peut également être représenté sous forme d'un diagramme d'activités, qui met l'accent sur la logique de décision plutôt que sur les échanges entre composants.

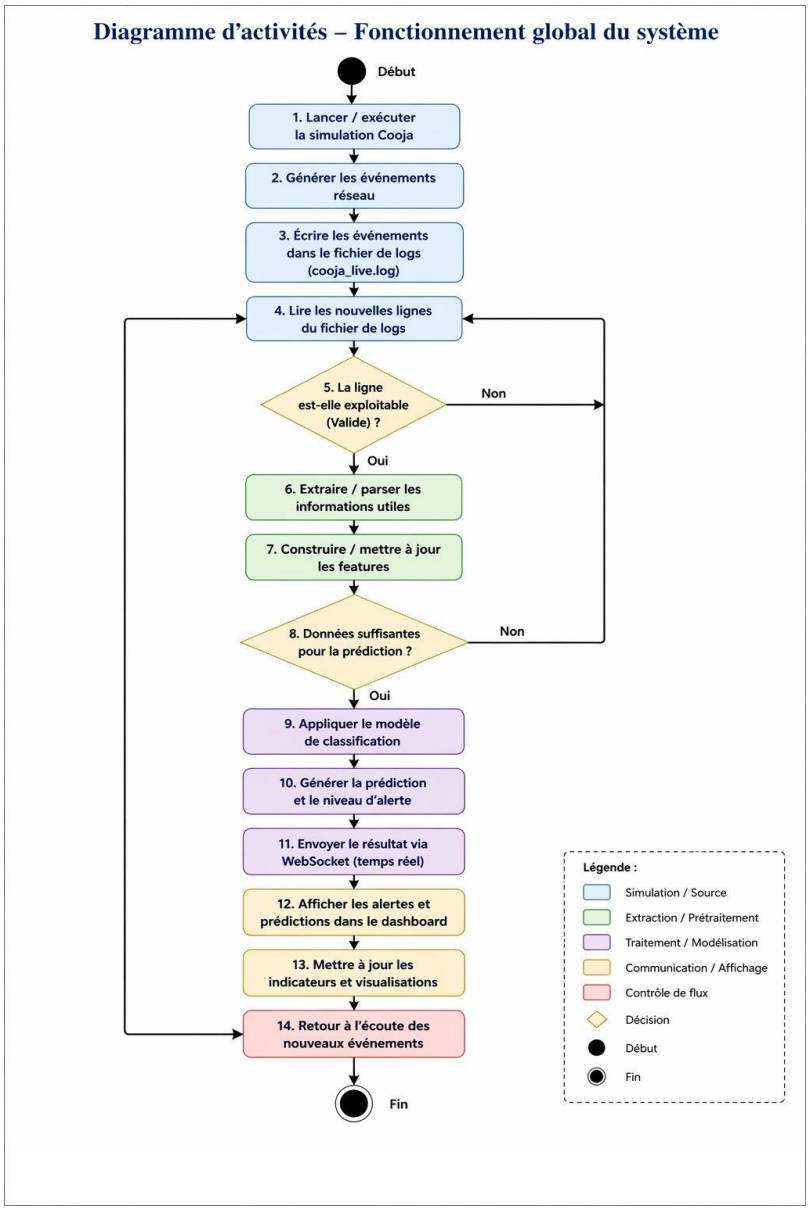


Figure 4.3 – Diagramme d'activités : Processus de détection et de supervision

Ce diagramme détaille les 14 étapes logiques du système lors du traitement des données. Il met en évidence deux points de décision critiques : la vérification de la validité de la ligne lue (étape 5) et la vérification de la suffisance des données pour déclencher une prédiction (étape 8). Ces contrôles garantissent la fiabilité des prédictions en évitant de classer des observations incomplètes ou corrompues.

4.6 Modélisation structurelle du système

Afin de compléter la présentation de l'architecture et du fonctionnement global, la conception du système est représentée par un diagramme de classes mettant en évidence les relations logiques entre les entités intervenant dans la chaîne de traitement.

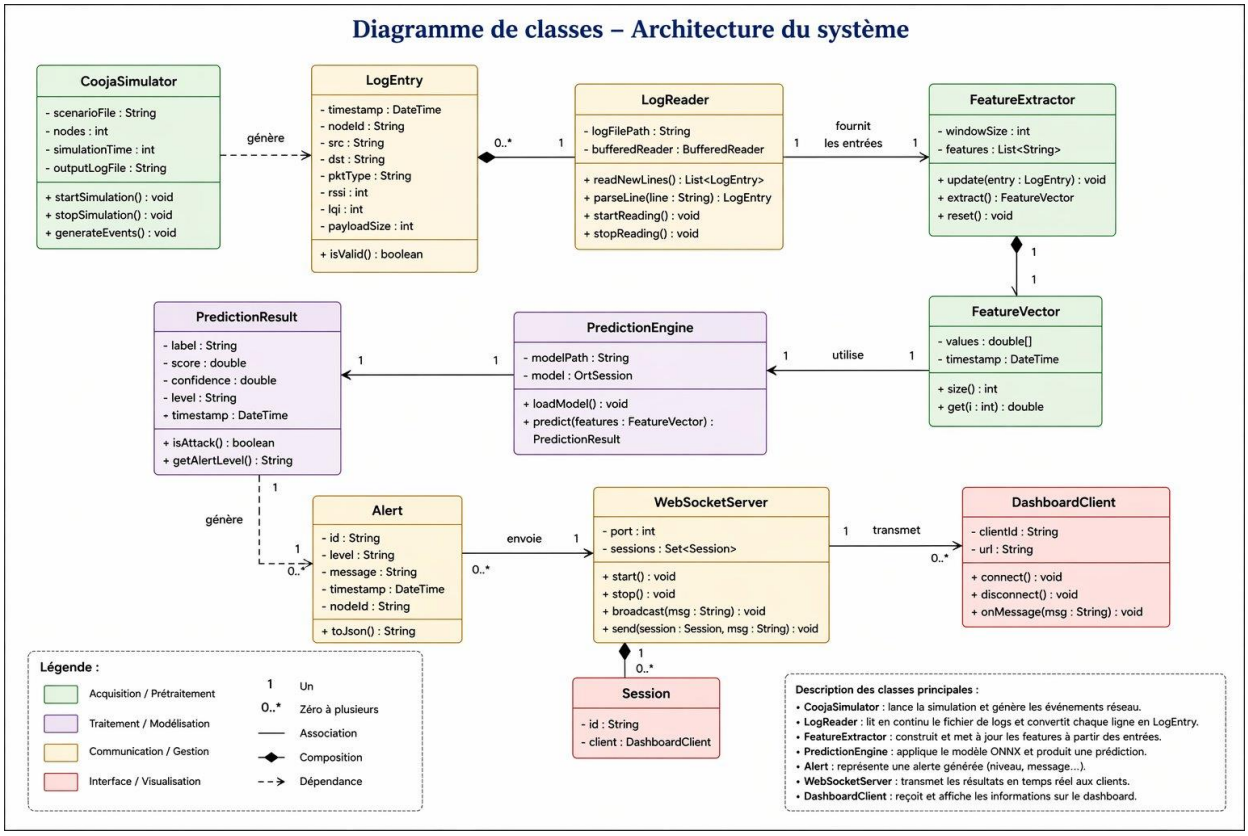


Figure 4.4 – Diagramme de classes : Architecture structurelle du système proposé

Ce diagramme identifie neuf classes principales organisées en quatre couches fonctionnelles. La couche acquisition regroupe CoojaSimulator et LogEntry, qui modélisent respectivement la source de simulation et les entrées parsées du fichier de logs. La couche prétraitement est assurée par LogReader et FeatureExtractor, qui lisent les logs en continu et construisent les vecteurs de features à partir d'une fenêtre glissante. La couche traitement comprend PredictionEngine et PredictionResult, qui appliquent le modèle ONNX et encapsulent les résultats de classification avec leur niveau d'alerte. Enfin, la couche communication et supervision regroupe Alert,

WebSocketServer, Session et DashboardClient, assurant la génération, la transmission et l'affichage des alertes en temps réel.

Les relations entre ces classes révèlent une architecture cohérente : CoojaSimulator génère des LogEntry, LogReader les fournit à FeatureExtractor qui construit un FeatureVector utilisé par PredictionEngine. PredictionResult génère des Alert transmises via WebSocketServer aux DashboardClient connectés.

4.7 Choix techniques de conception

Les choix techniques retenus ont été guidés par les besoins identifiés dans le chapitre précédent et par la nécessité de maintenir une architecture cohérente, modulaire et compatible avec une logique de détection live. Le tableau suivant synthétise ces choix et les justifie en regard des alternatives écartées.

Tableau 4.3 – Choix techniques et justifications

Choix technique	Justification	Alternative écartée
Cooja (simulateur)	Environnement IoT contrôlé, reproductible, sans matériel physique requis	Déploiement matériel réel (coûteux, difficile à reproduire)
Python (backend)	Écosystème ML riche, souplesse pour le parsing des logs et l'intégration du modèle	Java / Node.js (moins adapté à l'intégration ML)
ONNX (modèle ML)	Format portable et performant pour l'inférence, compatible Python	Pickle scikit-learn (moins portable, risques de compatibilité)
WebSocket	Transmission bidirectionnelle temps réel, faible latence, sans polling	REST API HTTP (polling périodique, latence plus élevée)
Dashboard séparé	Séparation claire backend / frontend, maintenabilité améliorée	Interface intégrée au backend (couplage fort, maintenance difficile)

Le recours à Cooja s'explique par sa capacité à simuler un environnement IoT contrôlé et reproductible. Le backend Python offre la souplesse nécessaire pour la lecture des logs, l'extraction des features et l'intégration du modèle ONNX. WebSocket garantit une transmission à faible latence vers le dashboard. La séparation du dashboard en composante distincte clarifie les responsabilités et améliore la maintenabilité de l'ensemble.

4.8 Conclusion

À travers ce chapitre, la conception générale du système proposé a permis de mettre en évidence l'architecture retenue pour assurer la détection d'attaques dans un environnement IoT simulé. La solution repose sur une chaîne cohérente reliant la simulation Cooja, l'exploitation du fichier cooja_live.log, le calcul des features par app.py, l'application du modèle de classification et la transmission des résultats vers le dashboard via WebSocket.

La description détaillée des composants (section 3.3), complétée par les cas d'utilisation (section 3.4), les diagrammes de séquence et d'activités (section 3.5) et le diagramme de classes (section 3.6), offre une vision complète de la solution, tant du point de vue dynamique que structurel. Les choix techniques justifiés en section 3.7 constituent un référentiel de conception solide pour la suite du travail, en particulier pour l'analyse approfondie du choix des features et la présentation de l'implémentation effective du système.

Chapitre 5 : Analyse et choix des features

Chapitre 5 : Analyse et choix des feautres

5.1 Introduction

Dans un système de détection d'intrusion fondé sur l'intelligence artificielle, la qualité de la détection dépend directement de la pertinence des variables fournies au modèle de classification. Ces variables, généralement appelées features, jouent un rôle central dans la représentation du comportement du réseau. Elles permettent de transformer les événements bruts observés dans les logs en informations quantitatives exploitables par le classifieur.

Dans le cadre de ce projet, l'analyse et le choix des features constituent une étape déterminante, car le modèle ne détecte pas directement une attaque à partir d'une simple lecture des logs. Il détecte plutôt des variations mesurables du comportement réseau, traduites sous forme de variables numériques. L'enjeu de ce chapitre est donc de montrer comment les features retenues permettent de capturer les effets des différentes attaques étudiées, et pourquoi elles représentent une base cohérente pour la détection.

L'approche adoptée repose sur l'idée que chaque attaque modifie le comportement normal du réseau selon une signature particulière. Certaines attaques augmentent fortement le volume d'envoi, d'autres dégradent la réception, introduisent des pertes, provoquent des retards anormaux ou créent des incohérences entre requêtes et réponses. Les features choisies ont précisément pour objectif de mesurer ces changements afin de permettre au modèle d'apprendre la signature de chaque type d'attaque.

5.2 Environnement de collecte des données

Afin d'illustrer l'origine des données exploitées dans ce travail, il est utile de présenter l'environnement Cooja dans lequel les traces réseau sont générées. La simulation permet d'observer les échanges entre les nœuds et de produire les logs à partir desquels les features sont calculées.

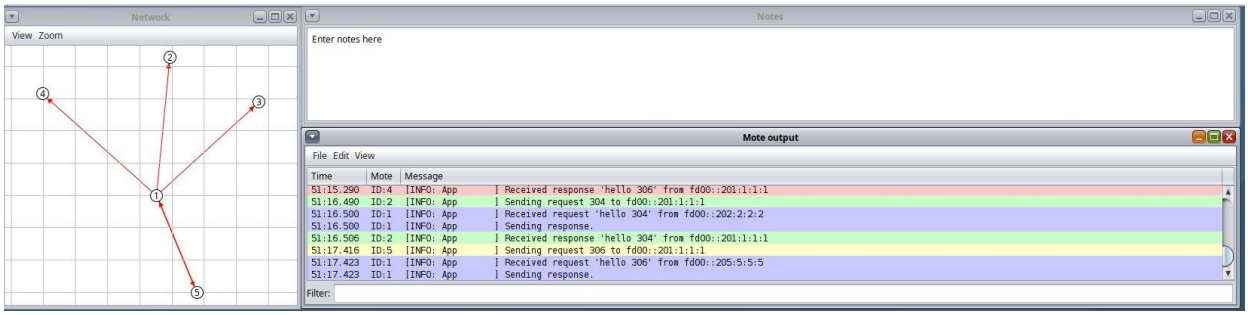


Figure 5.1 – Capture de l'environnement Cooja : topologie simulée et traces de communication

La Figure 4.1 montre un exemple d'exécution dans Cooja. La partie gauche représente la topologie du réseau simulé avec cinq nœuds communicants, tandis que la partie droite affiche le journal des messages échangés (Mote output). On peut y lire les requêtes envoyées, les réponses reçues, les horodatages et les identifiants des nœuds impliqués. Ces traces constituent la source principale des données exploitées pour calculer les features utilisées dans la détection.

À partir de ces événements, le système identifie les requêtes envoyées, les réponses reçues, les délais observés ainsi que les éventuelles pertes ou incohérences. Ces éléments servent de base au calcul des variables d'entrée du modèle de classification, selon le mécanisme de fenêtre glissante décrit en section 4.4.

4.3 Importance des features dans un système IDS

Dans un IDS fondé sur l'intelligence artificielle, les features représentent l'interface entre les données brutes du réseau et la décision produite par le modèle**[25], [26]**. Sans une sélection pertinente de ces variables, le classifieur ne peut pas distinguer correctement les comportements normaux des comportements malveillants. Les features ne constituent donc pas un simple détail technique, mais un élément central de la logique de détection.

Leur importance vient du fait qu'elles permettent de résumer le comportement du réseau sous une forme structurée. Au lieu de traiter directement l'ensemble des événements textuels issus des logs, le système extrait un ensemble limité de variables capables de représenter l'état du trafic, son évolution et ses anomalies. Cette transformation rend possible l'utilisation d'un modèle de classification supervisée, tout en améliorant la lisibilité de l'analyse.

Dans ce projet, les features sont organisées autour de cinq dimensions complémentaires du comportement réseau : l'envoi, la réception, la perte, le délai et la cohérence des échanges. Cette structuration permet de couvrir un spectre large de comportements anormaux et d'assurer que le modèle dispose d'informations suffisantes pour distinguer les différents types d'attaques.

5.4 Mécanisme de calcul : fenêtre glissante

Les features ne sont pas calculées à partir d'événements isolés, mais à partir d'une fenêtre glissante d'observations consécutives. Ce mécanisme est essentiel dans une architecture live, car il permet d'agréger plusieurs événements réseau avant de produire une prédiction, ce qui renforce la fiabilité et la robustesse de la détection.

Le principe de la fenêtre glissante consiste à maintenir un buffer des N derniers événements lus dans le fichier de logs. À chaque nouveau événement ajouté, l'événement le plus ancien est retiré du buffer, et l'ensemble des features est recalculé sur la fenêtre courante. Ce recalcul continu assure que chaque prédiction reflète l'état récent du réseau, sans nécessiter de relecture complète du fichier de logs **[27], [28]**.

Cette approche présente plusieurs avantages. Elle permet d'abord de lisser les fluctuations ponctuelles du trafic, en évitant que des événements isolés ne déclenchent de fausses alertes. Elle permet ensuite de capturer des tendances temporelles, comme une accélération progressive du rythme d'envoi ou une dégradation continue du taux de réception. Enfin, elle maintient la compatibilité avec une détection quasi-temps réel, car le recalcul des features est effectué de manière incrémentale**[29]**.

Tableau 5.1 – Paramètres de la fenêtre glissante et impact sur le calcul des features

Paramètre	Valeur / Description	Rôle dans le calcul des features
Taille de fenêtre	N événements consécutifs	Définit le nombre d'observations agrégées avant chaque prédiction
Pas de glissement	1 événement (fenêtre glissante)	Chaque nouvel événement déclenche un recalcul des features

sent_count	Comptage dans la fenêtre courante	Nombre de requêtes envoyées dans les N derniers événements
rate_sent	sent_count / durée de la fenêtre (s)	Fréquence d'envoi normalisée par le temps
loss	sent_count − recv_count dans la fenêtre	Différence entre émissions et réceptions dans la fenêtre
avg_delay	Moyenne des délais (réponse_ts − requête_ts)	Calculé uniquement sur les paires requête/réponse appariées
p95_delay	95e percentile des délais dans la fenêtre	Résistant aux valeurs moyennes trompeuses
matched_responses	Comptage des paires (req_id, resp_id) valides dans la fenêtre	Vérifie la cohérence logique des échanges

5.5 Présentation des features retenues

Le modèle de détection utilisé dans ce projet repose sur dix features principales, extraites à partir des événements observés dans les logs générés par Cooja. Ces variables sont regroupées selon cinq dimensions du comportement réseau, comme le synthétise le tableau suivant.

Tableau 5.2 – Regroupement des features par dimension

Dimension	Features associées	Comportement mesuré
Envoi	sent_count, rate_sent	Volume et rythme d'émission des requêtes
Réception	recv_count, rate_recv, recv_ratio	Capacité du réseau à retourner les réponses attendues
Perte	loss, loss_ratio	Quantité et proportion de messages non aboutis
Délai	avg_response_delay_ms,	Temps de réponse moyen et délais

	p95_response_delay_ms	extrêmes
Cohérence	matched_responses	Correspondance logique entre requêtes et réponses

5.5.1 Features liées à l'envoi

sent_count représente le nombre total de requêtes envoyées dans la fenêtre courante. Il est calculé par simple comptage des paquets émis identifiés dans les logs. Cette variable permet d'observer si le trafic devient anormalement élevé, ce qui constitue un indicateur précoce d'une attaque de type DoS.

rate_sent mesure la vitesse d'envoi des requêtes, exprimée en paquets par seconde. Elle est obtenue en divisant sent_count par la durée de la fenêtre temporelle. Cet indicateur est particulièrement sensible aux accélérations brutales du rythme d'émission, qui caractérisent les attaques par inondation.

5.5.2 Features liées à la réception

recv_count indique le nombre de réponses reçues dans la fenêtre. Combiné à sent_count, il permet d'évaluer si le réseau retourne correctement les réponses attendues.

rate_recv mesure la vitesse de réception des réponses en paquets par seconde. Une baisse significative de rate_recv par rapport à rate_sent signale une dégradation du flux retour.

recv_ratio correspond à la proportion de réponses reçues par rapport aux requêtes envoyées (recv_count / sent_count). Une valeur proche de 1 indique un réseau sain, tandis qu'une valeur faible signale souvent une attaque affectant la réception, comme blackhole ou selective.

5.5.3 Features liées à la perte

loss représente le nombre brut de messages perdus dans la fenêtre, calculé comme la différence entre sent_count et recv_count. Il fournit une mesure directe des paquets non aboutis.

loss_ratio exprime la proportion de messages perdus par rapport au volume total d'envoi (loss / sent_count). Cette variable normalisée est plus précise que la perte brute, car elle tient compte de l'intensité du flux observé et permet de comparer des fenêtres de tailles différentes.

5.5.4 Features liées au délai

avg_response_delay_ms mesure le délai moyen entre l'envoi d'une requête et la réception de la réponse correspondante, exprimé en millisecondes. Il est calculé sur les paires requête/réponse correctement appariées dans la fenêtre. Une hausse anormale de cette valeur caractérise les attaques de type Delay.

p95_response_delay_ms correspond au 95e percentile des délais observés dans la fenêtre. Cet indicateur est plus robuste que la moyenne, car il met en évidence les réponses particulièrement lentes que la moyenne pourrait masquer en cas de distribution asymétrique.

5.5.5 Feature liée à la cohérence

matched_responses compte le nombre de réponses correctement associées à leurs requêtes d'origine dans la fenêtre courante. Cette feature évalue la cohérence logique des échanges et est particulièrement utile pour détecter les attaques de type Replay, qui perturbent la relation normale entre requête et réponse en rejouant d'anciens messages.

Tableau 5.3 – Features retenues, mode de calcul et intérêt pour la détection

Feature	Signification	Dimension	Calcul	Intérêt pour la détection
sent_count	Nombre de requêtes envoyées dans la fenêtre	Envoi	Comptage des paquets émis	Détecter une hausse anormale du trafic (DoS)
rate_sent	Vitesse d'envoi des requêtes (paquets/seconde)	Envoi	sent_count / durée_fenêtre	Identifier une accélération brutale du rythme d'émission
recv_count	Nombre de réponses reçues dans la fenêtre	Réception	Comptage des paquets reçus	Observer une baisse anormale de réception

rate_recv	Vitesse de réception des réponses (paquets/seconde)	Réception	recv_count / durée_fenêtre	Mesurer la dégradation du flux retour
recv_ratio	Proportion de réponses reçues / requêtes envoyées	Réception	recv_count / sent_count	Détecter une anomalie de réception (blackhole, selective)
loss	Nombre de messages perdus dans la fenêtre	Perte	sent_count − recv_count	Détecter les pertes directes de paquets
loss_ratio	Proportion de pertes / volume total	Perte	loss / sent_count	Mesure normalisée, plus précise que loss brute
avg_response_delay_ms	Délai moyen requête → réponse (ms)	Délai	Moyenne des délais mesurés	Détecter des ralentissements anormaux (Delay)
p95_response_delay_ms	95e percentile des délais observés (ms)	Délai	Percentile 95 des délais	Mettre en évidence les délais extrêmes masqués par la moyenne
matched_responses	Réponses correctement associées aux requêtes	Cohérence	Comptage des paires requête/réponse valides	Détecter incohérences et répétitions (Replay)

5.6 Rôle des features dans la détection des attaques

L'intérêt des features retenues ne réside pas uniquement dans leur signification individuelle, mais dans leur capacité collective à représenter les modifications introduites par chaque type d'attaque. En effet, chaque attaque laisse une empreinte particulière dans les données, et c'est la

combinaison de plusieurs features qui permet au modèle de reconnaître une signature de comportement spécifique.

Les features liées à l'envoi (sent_count, rate_sent) jouent un rôle déterminant dans l'identification des attaques provoquant une hausse anormale du trafic. Les features de réception et de perte (recv_count, recv_ratio, loss, loss_ratio) deviennent prioritaires lorsque l'attaque réduit le nombre de réponses ou augmente les pertes. Les variables de délai (avg_response_delay_ms, p95_response_delay_ms) sont essentielles pour les scénarios où la communication reste présente mais devient anormalement lente. Enfin, matched_responses permet de repérer des attaques plus subtiles qui perturbent la structure logique des échanges sans nécessairement modifier les volumes.

Cette complémentarité entre les cinq dimensions assure une couverture large des comportements anormaux possibles. Une attaque ne se manifeste pas toujours par une seule variable isolée — c'est la relation entre plusieurs indicateurs qui permet au système de reconnaître une signature particulière. Cette logique justifie le choix d'un ensemble de features complémentaires plutôt qu'une simple variable unique de surveillance.

5.7 Lien entre features et types d'attaques

L'analyse des liens entre attaques et features constitue un point central de ce chapitre. Elle permet de justifier le choix des variables en montrant que chacune d'elles répond à un comportement réseau identifiable. Le tableau suivant synthétise ces relations pour les neuf comportements étudiés.

Tableau 5.4 – Lien entre types d'attaques et features dominantes

Type d'attaque	Features les plus liées	Effet observé sur le réseau
Normal	sent_count, recv_count, recv_ratio, loss_ratio, avg_response_delay_ms	Trafic stable, perte faible, délai dans la plage normale
DoS	sent_count, rate_sent, loss, recv_ratio	Hausse forte du volume et du rythme

		d'envoi, saturation possible
Delay	avg_response_delay_ms, p95_response_delay_ms	Réponses lentes malgré la présence du trafic
Replay	matched_responses, recv_count, avg_response_delay_ms	Incohérences ou répétitions dans les échanges requête/réponse
Selective	recv_count, loss, loss_ratio, recv_ratio	Perte partielle des messages, réception dégradée
Blackhole	recv_count, loss, loss_ratio, recv_ratio	Forte chute de réception et pertes très élevées
Forwarding	recv_count, loss_ratio, recv_ratio, loss	Mauvais transfert des messages, effet proche d'une perte partielle
Routing selective	recv_count, loss_ratio, recv_ratio	Perte partielle sur le chemin multi-hop
Routing blackhole	recv_count, loss, loss_ratio, recv_ratio	Forte perte causée par un nœud de routage absorbant les paquets

5.7.1 Comportement normal

Dans un fonctionnement normal, le trafic reste relativement stable. Les valeurs de sent_count, recv_count, recv_ratio, loss_ratio et avg_response_delay_ms demeurent cohérentes avec un réseau sain : le taux de réception est élevé, la perte est faible et le délai reste dans une plage attendue.

5.7.2 Attaque DoS

L'attaque DoS se caractérise par un envoi massif de requêtes en peu de temps. Les features sent_count et rate_sent augmentent fortement et brutalement. Cette attaque peut également dégrader recv_ratio et augmenter loss si le réseau ou la cible se retrouve saturé. C'est l'une des attaques les plus faciles à détecter grâce aux variables d'envoi.

5.7.3 Attaque Delay

L'attaque Delay ne bloque pas la communication mais retarde les réponses. Les features avg_response_delay_ms et p95_response_delay_ms augmentent significativement, tandis que les volumes d'envoi et de réception peuvent rester normaux. Le p95 est particulièrement utile ici, car il révèle des délais extrêmes que la moyenne pourrait atténuer.

5.7.4 Attaques de type perte (Selective, Blackhole, Forwarding, Routing)

Ces quatre types d'attaques se manifestent principalement par une dégradation des features de réception et de perte. Le blackhole provoque une chute quasi-totale de recv_count et une forte hausse de loss_ratio. Le selective et le routing selective produisent une perte partielle, avec recv_ratio et loss_ratio anormaux mais non nuls. Le forwarding génère un effet similaire au selective, mais lié à un mauvais transfert au niveau d'un nœud intermédiaire.

5.7.5 Attaque Replay

L'attaque Replay rejouant d'anciens messages, elle perturbe la cohérence logique des échanges. La feature matched_responses devient anormale, et recv_count peut augmenter artificiellement en raison des réponses répétées. avg_response_delay_ms peut également être affecté si les réponses rejouées arrivent avec des décalages temporels.

5.8 Justification du choix des features

Le choix des features retenues repose sur quatre critères principaux, qui ont guidé la sélection tout au long du projet.

Le premier critère concerne la couverture des cinq dimensions du comportement réseau. Les variables choisies permettent d'observer l'envoi, la réception, la perte, le délai et la cohérence, assurant ainsi une vision complète du trafic et évitant de limiter la détection à un seul type d'anomalie.

Le deuxième critère repose sur le lien explicite entre certaines attaques et certaines variables, comme l'illustre le Tableau 4.4. Chaque attaque possède une signature reconnaissable dans au moins deux features, ce qui garantit que le modèle dispose d'informations discriminantes pour chaque scénario.

Le troisième critère concerne la calculabilité dans une architecture live. Toutes les features peuvent être calculées incrementalement à partir des logs générés par Cooja, en utilisant le mé-

canisme de fenêtre glissante décrit en section 4.4. Elles ne nécessitent pas d'accès à des données historiques volumineuses, ce qui les rend compatibles avec le traitement quasi-temps réel.

Le quatrième critère est la simplicité d'interprétation. Les features choisies restent compréhensibles du point de vue réseau et peuvent être expliquées clairement dans un contexte académique ou opérationnel, ce qui renforce la cohérence entre la logique technique du système et sa présentation.

5.9 Conclusion

L'analyse présentée dans ce chapitre montre que le choix des features constitue un élément central de la solution de détection proposée. Les dix variables retenues permettent de représenter cinq dimensions majeures du comportement réseau — l'envoi, la réception, la perte, le délai et la cohérence — offrant ainsi une base cohérente pour l'application du modèle de classification.

Le mécanisme de fenêtre glissante présenté en section 4.4 assure que ces features sont calculées de manière continue et incrémentale à partir des logs générés par Cooja, garantissant la compatibilité avec l'architecture live du système. L'étude du lien entre attaques et features (section 4.7 et Tableau 4.4) a confirmé que chaque type d'attaque laisse une signature identifiable dans les données, justifiant ainsi la pertinence des variables sélectionnées.

Ce chapitre prépare naturellement le chapitre suivant, consacré à la mise en œuvre effective de la solution : l'implémentation du pipeline de détection, les tests réalisés dans l'environnement simulé et l'analyse des résultats obtenus.

Chapitre 6 : Mise en œuvre, validation et résultats expérimentaux

Chapitre 6 : Mise en œuvre, validation et résultat expérimentaux

6.1 Introduction

Les chapitres précédents ont posé les fondements théoriques et conceptuels du système proposé. L'architecture générale a été définie au chapitre 3, articulant les composants du pipeline de détection autour d'une logique live cohérente. Le chapitre 4 a ensuite justifié le choix des dix features retenues pour représenter le comportement du réseau IoT, en établissant leur lien avec les différents types d'attaques étudiés. Il convient à présent de décrire la concrétisation effective de ces travaux, en exposant l'implémentation du système, les expérimentations conduites et les résultats obtenus [30].

Ce chapitre s'organise autour de plusieurs axes complémentaires. Il commence par rappeler l'architecture globale du système avant de décrire l'environnement de développement retenu. Il présente ensuite les étapes de construction du dataset à partir des simulations Cooja, l'implémentation du pipeline live de détection, puis l'interface de supervision développée. La partie centrale est consacrée à la validation expérimentale du système, avec une analyse détaillée des résultats obtenus pour chaque type d'attaque et une lecture approfondie de la matrice de confusion. Une discussion critique clôt ce chapitre en identifiant les forces du système, ses limites, et les perspectives d'amélioration envisageables pour des travaux futurs.

6.2 Architecture globale du système

Le système de détection d'intrusion développé dans ce projet repose sur une architecture pipeline orientée live, dans laquelle les données circulent de manière continue depuis l'environnement de simulation jusqu'à l'interface de supervision, sans nécessiter d'interruption ni de traitement par lots. Cette organisation a été choisie pour refléter les exigences d'un système de détection appliqué à un environnement IoT dynamique, dans lequel les événements réseau se produisent en flux continu et doivent être analysés au fur et à mesure de leur apparition.

Le point d'entrée du système est l'environnement Cooja, qui simule un réseau de capteurs IoT et génère les événements réseau observés lors de l'exécution de différents scénarios, qu'il s'agisse de trafic normal ou de scénarios d'attaques. Ces événements sont enregistrés en temps réel dans le fichier cooja_live.log, qui constitue la source de données brutes de l'ensemble du pipeline. Le backend Python, orchestré par les scripts app.py et live_predict_tail.py, surveille ce fichier en continu, valide chaque nouvelle ligne, calcule les features sur une fenêtre glissante et soumet

le vecteur résultant au modèle Random Forest pour classification. La prédiction produite, accompagnée d'un score de confiance et d'un niveau d'alerte, est immédiatement transmise via WebSocket au dashboard de supervision, qui l'affiche en temps réel à l'intention de l'administrateur réseau.

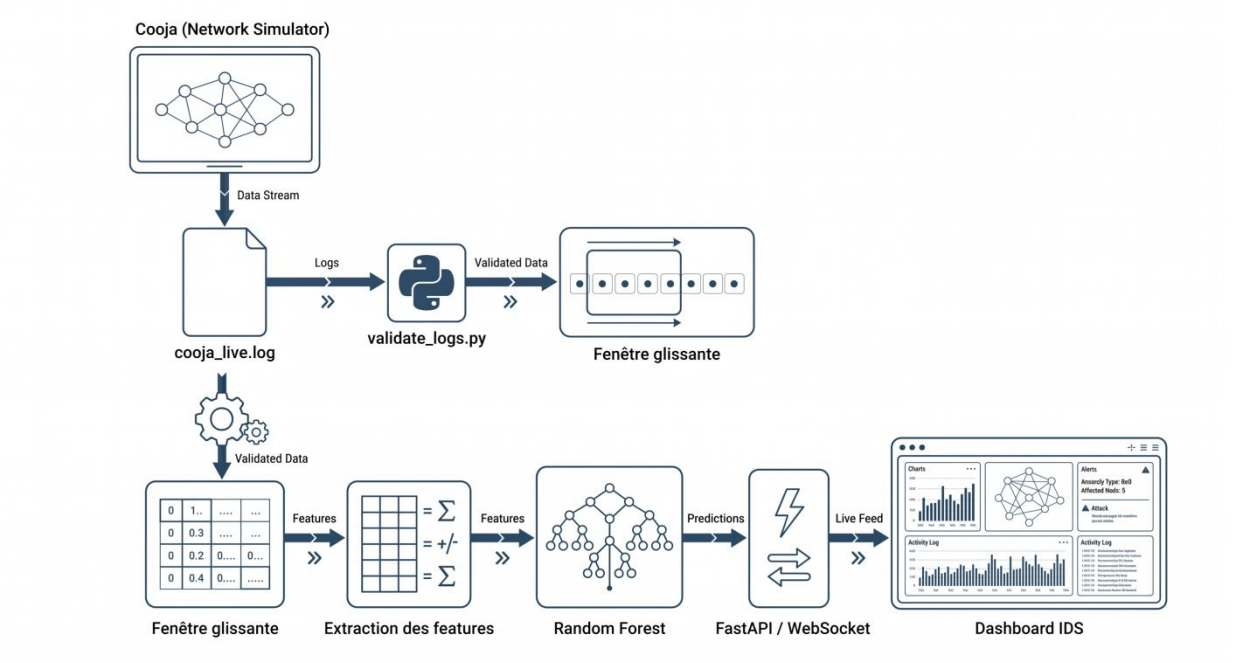


Figure 6.1 – Architecture globale du système IDS live : de la simulation Cooja au dashboard de supervision

Cette organisation modulaire présente plusieurs avantages. Elle permet d'abord de séparer clairement les responsabilités de chaque composant, facilitant ainsi la maintenance et l'évolution du système. Elle garantit ensuite que chaque prédiction est produite dans un délai minimal après l'apparition de l'événement réseau correspondant, ce qui est essentiel pour une supervision efficace. Enfin, elle assure la cohérence technique entre la simulation, le traitement et la visualisation, en maintenant un flux de données structuré et traçable à chaque étape du pipeline.

6.3 Environnement de développement et organisation du projet

Le développement du système a été réalisé sous Linux Ubuntu 22.04 LTS, un environnement particulièrement adapté au projet en raison de sa compatibilité native avec Cooja et Contiki-

NG, ainsi que de la richesse des bibliothèques Python disponibles pour le traitement des données et l'apprentissage automatique. L'ensemble des composants du système — simulation, backend, modèle et interface — ont été développés et testés sur cette même plateforme, ce qui garantit la cohérence de l'environnement d'exécution.

Le backend du système repose sur Python 3.10, dont l'écosystème offre une combinaison optimale de bibliothèques pour le traitement des logs (lecture asynchrone, parsing), l'extraction des features (calcul numérique, fenêtre glissante) et l'inférence du modèle (scikit-learn). Le framework FastAPI a été retenu pour exposer les endpoints de l'API backend et gérer les connexions WebSocket avec le dashboard. Son déploiement s'appuie sur le serveur ASGI Uvicorn, qui assure un traitement asynchrone efficace des requêtes entrantes et des flux WebSocket simultanés, sans bloquer l'exécution du pipeline de détection. Le Tableau 5.1 synthétise l'ensemble des technologies utilisées.

Tableau 6.1 – Environnement de développement et outils utilisés

Composant	Outil / Version	Rôle dans le système
Système d'exploitation	Linux Ubuntu 22.04 LTS	Environnement d'exécution principal
Simulateur IoT	Cooja / Contiki-NG	Simulation du réseau IoT et génération des logs
Langage backend	Python 3.10	Traitement des logs, extraction des features, inférence
Framework web	FastAPI + Uvicorn	Serveur backend, API REST et WebSocket
Modèle ML	Random Forest (scikit-learn)	Classification du trafic réseau en 9 classes
Communication	WebSocket (asyncio)	Transmission temps réel backend → dashboard
Interface de supervision	HTML / CSS / JavaScript	Dashboard de visualisation des alertes
Scripts principaux	app.py, live_predict_tail.py, validate_logs.py	Pipeline complet de détection live
Fichier de logs	cooja_live.log	Source de données brutes générée par Cooja

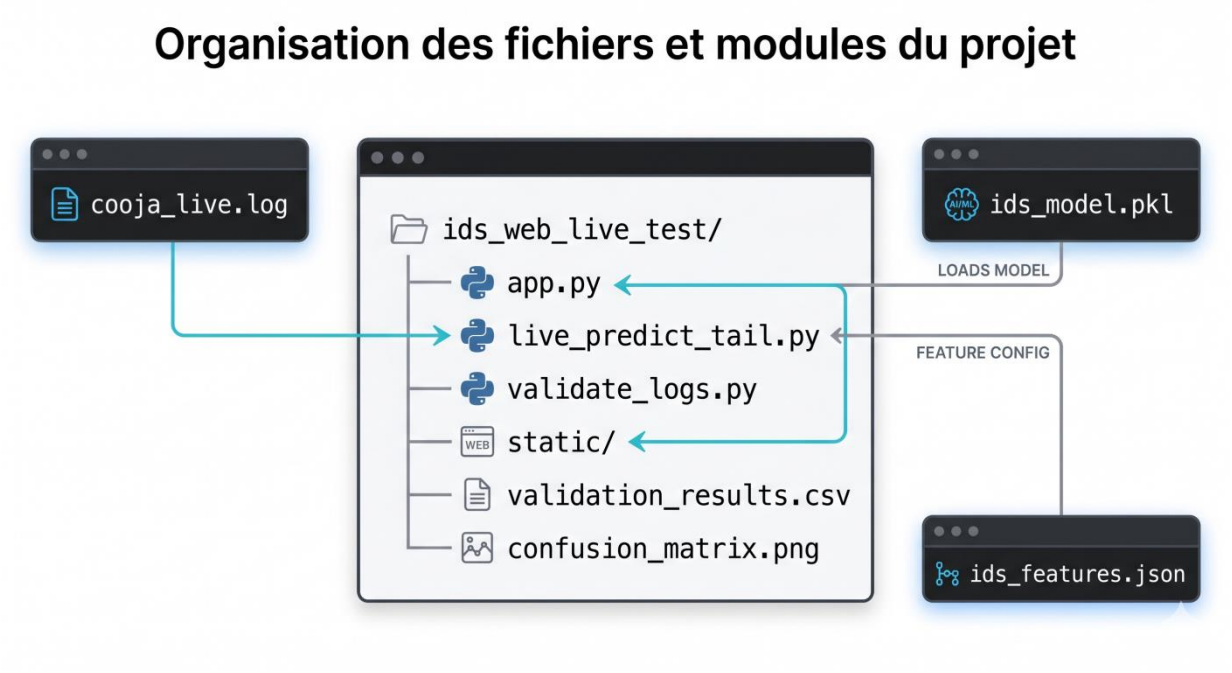


Figure 6.2 – Organisation des fichiers et modules du projet

L'interface backend FastAPI, accessible localement sur le port 8000, expose les routes nécessaires à la communication avec le dashboard et gère les connexions WebSocket entrantes. La Figure 5.3 illustre le démarrage du serveur Uvicorn sur la machine de développement, confirmant que le backend est actif, que le rechargement automatique est activé (--reload) et que le serveur est en attente de connexions sur http://127.0.0.1:8000.

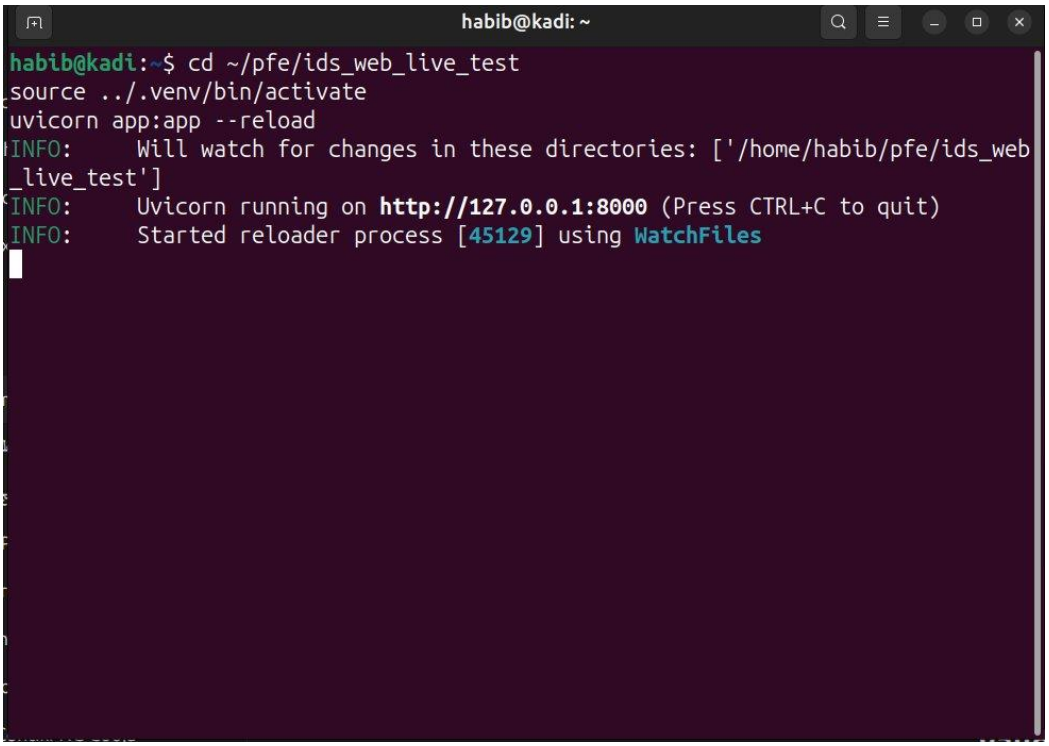


Figure 6.3 – Démarrage du serveur backend FastAPI avec Uvicorn sur http://127.0.0.1:8000

La structure du projet est organisée de manière modulaire, avec une séparation claire entre les scripts de traitement (live_predict_tail.py, validate_logs.py), le serveur backend (app.py), le modèle entraîné et les fichiers de l'interface web. Cette organisation facilite la maintenance indépendante de chaque composant et permet d'intervenir sur un module sans affecter les autres parties du système.

6.4 Génération du dataset et collecte des logs

La construction du dataset constitue une étape fondamentale du projet, car la qualité et la représentativité des données d'entraînement conditionnent directement les performances du modèle de classification. Dans le cadre de ce travail, le dataset a été entièrement construit à partir de simulations réalisées dans l'environnement Cooja, en reproduisant manuellement neuf scénarios distincts : un comportement réseau normal et huit types d'attaques IoT couvrant les principales

menaces identifiées dans les réseaux de capteurs sans fil (DoS, Blackhole, Delay, Replay, Forwarding, Routing Blackhole, Routing Selective et Selective) **[31]**.

Pour chaque scénario, une topologie réseau composée de plusieurs nœuds communicants a été configurée dans Cooja. Les nœuds échangent des messages selon un protocole applicatif simple — requête/réponse — reproduisant les interactions caractéristiques d'un réseau de capteurs IoT. L'attaque est ensuite injectée en modifiant le comportement d'un ou plusieurs nœuds du réseau, selon la nature du scénario simulé. Cooja enregistre l'intégralité des événements réseau produits pendant la simulation dans le fichier cooja_live.log, qui constitue la source brute des données.

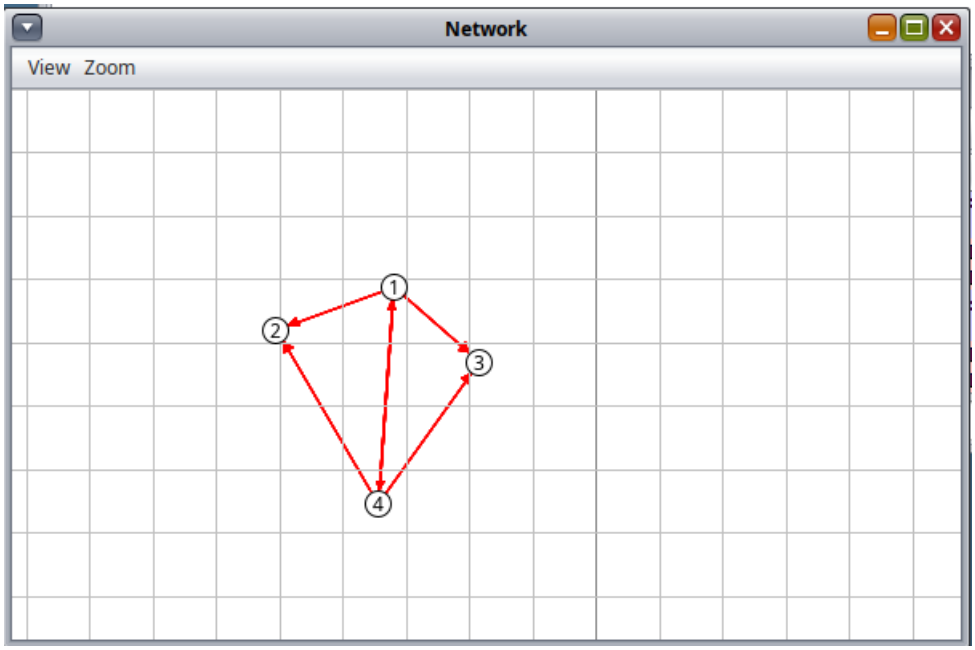


Figure 6.4 – Topologie réseau simulée dans l'environnement Cooja

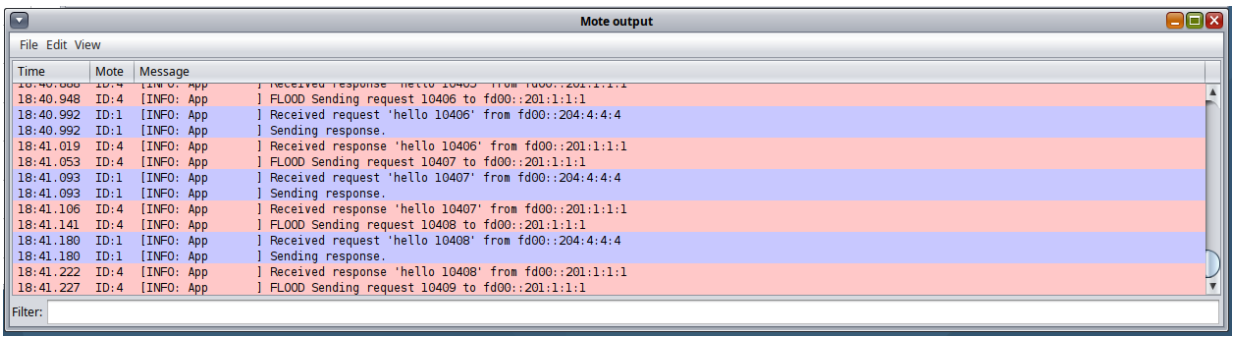


Figure 6.5 – Extrait du fichier cooja_live.log montrant les échanges réseau enregistrés

Les logs bruts ainsi collectés ont ensuite été traités par le script validate_logs.py, dont le rôle est de parser chaque ligne, de vérifier la présence et la cohérence des champs requis (horodatage, identifiant source, identifiant destination, type de message, RSSI, LQI, taille du paquet) et d'écarter les lignes invalides ou incomplètes. Ce filtrage préalable est essentiel pour garantir que seules des observations exploitables alimentent le pipeline d'extraction des features, évitant ainsi l'introduction de bruit dans le dataset.

Les événements validés sont ensuite agrégés par fenêtre glissante pour produire les vecteurs de features correspondant à chaque observation du dataset, puis labellisés avec le type de trafic simulé. Le dataset final a été divisé en deux sous-ensembles distincts : 80 % des observations ont été affectées à l'entraînement du modèle, les 20 % restants étant réservés pour le jeu de validation. Cette partition a été réalisée de manière stratifiée afin de préserver la distribution des classes dans chaque sous-ensemble et de garantir une évaluation représentative des performances du modèle.

6.5 Implémentation du pipeline live de détection

L'implémentation du pipeline live constitue le cœur opérationnel du système. Elle repose sur deux scripts principaux — live_predict_tail.py et validate_logs.py — qui orchestrent l'ensemble des opérations depuis la lecture continue des logs jusqu'à la transmission des prédictions au dashboard. Ce pipeline a été conçu pour s'exécuter de manière asynchrone, permettant de gérer simultanément la surveillance du fichier de logs, le calcul des features et la communication WebSocket sans qu'aucun de ces processus ne bloque les autres.

6.5.1 Lecture continue des logs — live_predict_tail.py

Le script live_predict_tail.py surveille le fichier cooja_live.log en mode lecture continue, en ouvrant le fichier en fin de contenu et en détectant chaque nouvelle ligne au fur et à mesure de son apparition, selon un mécanisme analogue à la commande Unix tail -f. Cette approche garantit une réactivité maximale du système : dès qu'un nouvel événement réseau est enregistré par Cooja, il est immédiatement pris en charge par le pipeline de traitement, sans délai d'attente

ou de scrutation périodique coûteuse. Chaque nouvelle ligne est transmise au module de validation pour vérification, puis ajoutée au buffer de la fenêtre glissante si elle est valide.

```
=== WINDOW RESULT ===
Window: 196548.797 -> 196553.797
Features: {'sent_count': 2.0, 'recv_count': 0.0, 'loss': 2.0, 'rate_sent': 0.4,
'rate_recv': 0.0, 'loss_ratio': 1.0, 'recv_ratio': 0.0, 'avg_response_delay_ms':
 0.0, 'p95_response_delay_ms': 0.0, 'matched_responses': 0.0}
Prediction: blackhole
Confidence: 0.7987
```

Figure 6.6 – Exécution de live_predict_tail.py : détection live dans le terminal

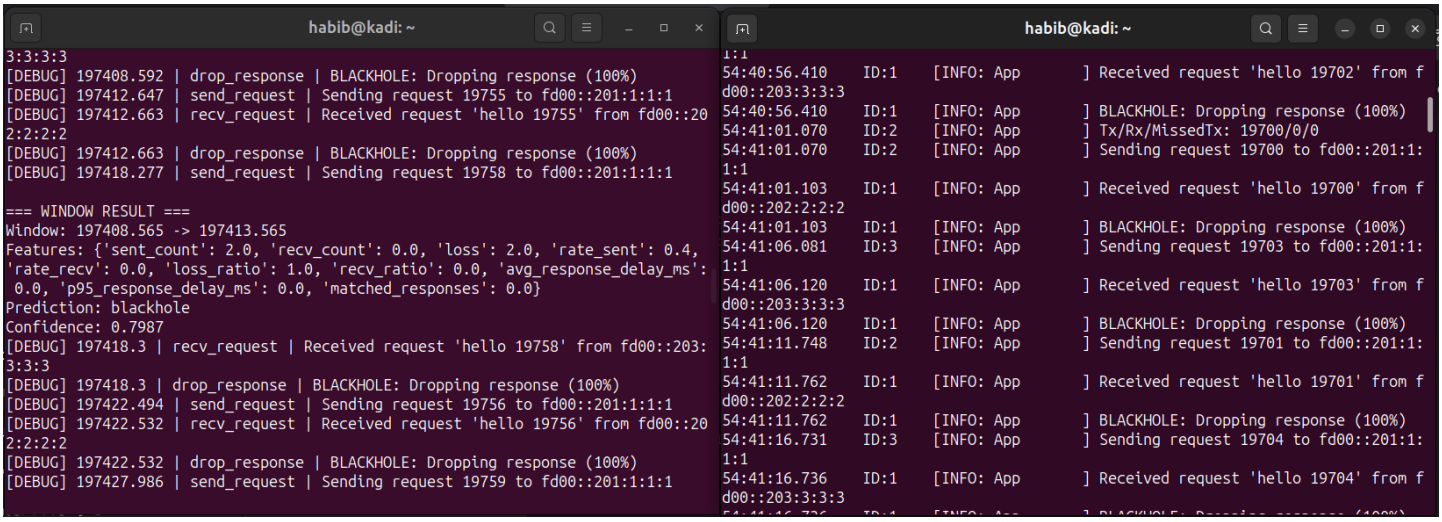


Figure 6.7 – Pipeline live complet : de la lecture des logs à la génération de la prédiction

6.5.2 Fenêtre glissante et extraction des features

Le mécanisme de fenêtre glissante joue un rôle central dans la robustesse de la détection. Plutôt que de classifier chaque événement réseau isolément — ce qui serait particulièrement sensible au bruit et aux fluctuations ponctuelles du trafic — le système agrège les N derniers événements valides dans un buffer circulaire. À chaque nouvel ajout, les dix features sont recalculées sur l'ensemble de la fenêtre courante : les features de comptage (sent_count, recv_count, loss, matched_responses) par simple agrégation, les features de ratio (rate_sent, rate_recv,

recv_ratio, loss_ratio) par normalisation, et les features de délai (avg_response_delay_ms, p95_response_delay_ms) à partir des paires requête/réponse appariées dans la fenêtre.

Ce mécanisme offre un équilibre fondamental entre réactivité et robustesse. Une fenêtre trop petite rend le système sensible aux variations ponctuelles et peut générer des faux positifs, tandis qu'une fenêtre trop grande retarde la détection et peut diluer le signal d'une attaque naissante dans le bruit de fond du trafic normal. La taille de la fenêtre a été calibrée empiriquement lors de la phase d'expérimentation afin d'optimiser ce compromis pour chaque type d'attaque.

6.5.3 Intégration du modèle Random Forest

Le modèle Random Forest a été entraîné hors ligne avec scikit-learn sur le dataset construit à partir des simulations Cooja. L'algorithme Random Forest a été retenu pour ses performances reconnues sur les données tabulaires, sa robustesse au bruit, sa capacité à gérer des classes multiples sans transformation préalable des données et sa facilité d'interprétation via l'analyse de l'importance des features. Une fois entraîné, le modèle est chargé en mémoire au démarrage d'app.py et reste disponible pour l'ensemble des requêtes d'inférence sans rechargement [33].

Pour chaque vecteur de features produit par le pipeline live, le modèle produit deux sorties : la classe prédite parmi les neuf comportements possibles et le vecteur de probabilités associé, dont la valeur maximale constitue le score de confiance. Ce score est particulièrement utile pour évaluer la certitude du modèle et pour déclencher des alertes proportionnées à la sévérité estimée de la menace détectée.

6.6 Interface de supervision — Dashboard IoT IDS

L'interface de supervision constitue le point de contact entre le système de détection et l'administrateur réseau. Développée en HTML, CSS et JavaScript, elle se connecte automatiquement au serveur WebSocket d'app.py à son chargement et reçoit en flux continu les messages JSON contenant les prédictions, scores de confiance et niveaux d'alerte. Cette architecture de communication garantit que le dashboard est toujours synchronisé avec l'état courant du système de détection, sans nécessiter de rechargement manuel ni de mécanisme de polling.

6,6.1 Composants et fonctionnalité

Le dashboard présente plusieurs composants complémentaires répartis sur une interface unique et épurée. En haut de page, quatre indicateurs synthétiques affichent en temps réel le nombre total d'alertes déclenchées depuis le début de la session, répartis par niveau de sévérité : CRITICAL, HIGH et MEDIUM/LOW. Ces indicateurs permettent à l'administrateur d'évaluer en un coup d'œil l'état général du réseau supervisé et d'identifier rapidement si une intervention est nécessaire.

La section centrale présente trois informations clés : l'attaque la plus fréquemment détectée dans la session courante (Top Attack), la dernière prédiction produite par le modèle (Current Prediction) accompagnée de son score de confiance, et la dernière alerte générée avec son horodatage précis. Les notifications d'alertes apparaissent en temps réel dans le coin supérieur droit de l'interface, permettant à l'opérateur de suivre l'activité du réseau sans avoir à consulter constamment le tableau principal. Une barre de recherche et un filtre de sévérité permettent de naviguer dans l'historique des alertes selon les besoins de l'analyse.

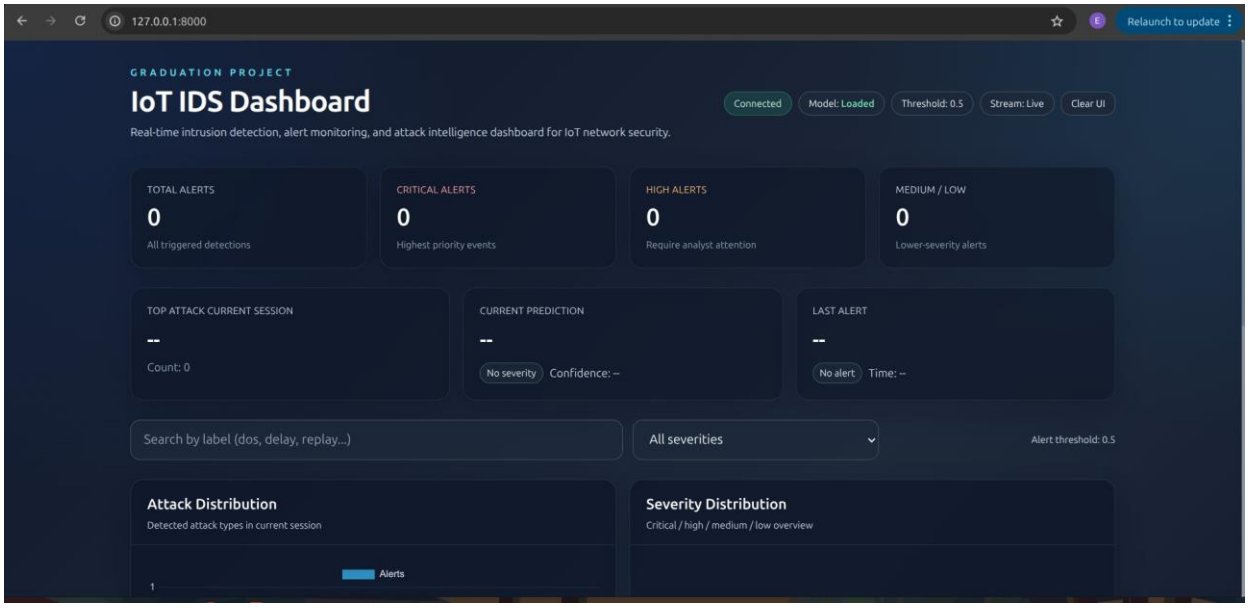


Figure 6.8 – Dashboard IoT IDS en mode normal : système connecté, modèle chargé, aucune alerte active

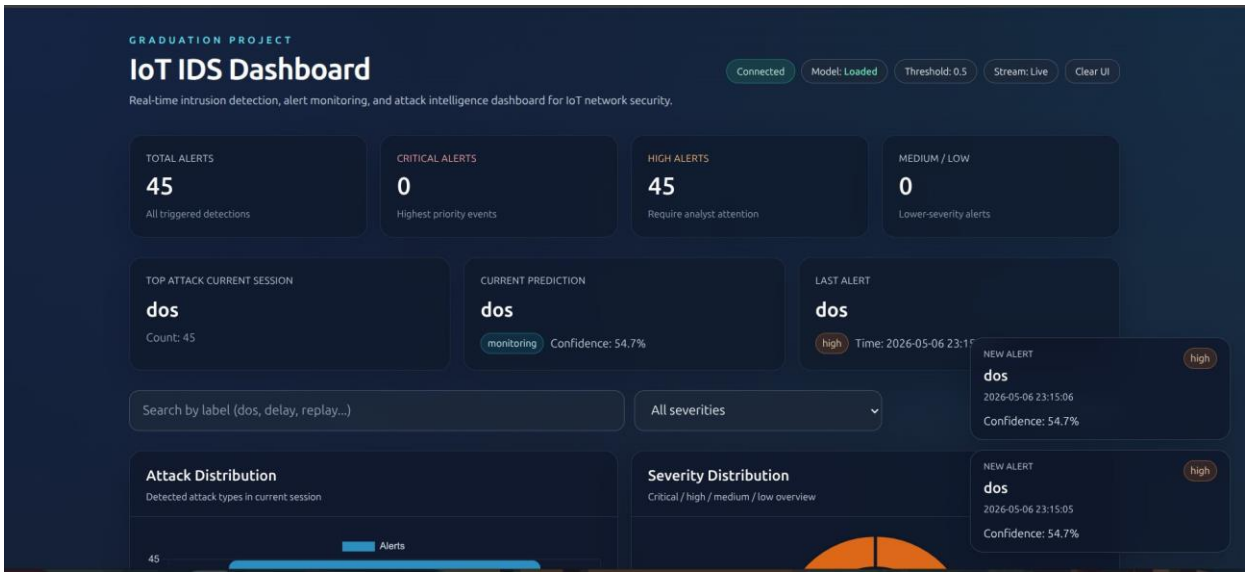


Figure 6.9 – Dashboard lors d'une attaque DoS détectée : 45 alertes HIGH, confiance 54.7 %

6.6.2 Intérêt opérationnel

L'intérêt opérationnel du dashboard réside dans sa capacité à rendre les résultats du modèle directement exploitables par un administrateur réseau, sans nécessiter de compétences spécifiques en apprentissage automatique. L'affichage du score de confiance en complément de la classe prédite est particulièrement utile : il permet à l'opérateur d'évaluer la certitude du système et de pondérer ses décisions en conséquence. Comme illustré par la Figure 5.9, lors de la détection d'une attaque DoS, le dashboard affiche clairement le type d'attaque détecté, le niveau de sévérité (HIGH), l'horodatage de la dernière alerte et le score de confiance (54.7 %), permettant une réaction rapide et informée.

L'indicateur de statut en haut à droite (Connected / Model: Loaded / Stream: Live) fournit en permanence une information transparente sur l'état du système : connexion WebSocket active, modèle correctement chargé et flux de données en cours. Cette transparence sur le fonctionnement interne contribue à la confiance de l'opérateur dans les alertes générées, en lui permettant de distinguer une absence d'alertes due à l'absence réelle de menaces d'une absence due à un dysfonctionnement du système.

6.7 Validation expérimentale et résultats

La phase de validation expérimentale vise à évaluer les performances du système de détection dans des conditions contrôlées, en utilisant des fichiers de logs dont le label est connu avec certitude. Cette évaluation permet de mesurer objectivement la capacité du modèle à classifier correctement chaque type de trafic, d'identifier les classes les mieux détectées, de mettre en évidence les ambiguïtés entre classes proches et de calculer les métriques de performance globales du système.

6.7.1 Protocole de validation

La validation du système a été conduite à l'aide du script validate_logs.py, qui applique le pipeline complet de détection sur des fichiers de logs pré-enregistrés correspondant à chaque type de trafic. Pour chaque fichier, le script extrait l'ensemble des fenêtres glissantes possibles, produit une prédiction pour chacune d'elles, puis détermine la prédiction majoritaire sur l'ensemble du fichier et calcule la confiance moyenne associée. La prédiction majoritaire est ensuite comparée à la classe attendue pour déterminer si le fichier est correctement classifié.

Au total, 34 239 prédictions ont été produites sur l'ensemble des neuf fichiers de validation, couvrant les neuf types de comportements réseau étudiés. La Figure 5.10 présente les résultats bruts affichés dans le terminal lors de l'exécution du script de validation, permettant de lire directement pour chaque classe la prédiction obtenue, la confiance moyenne et le nombre d'observations traitées.

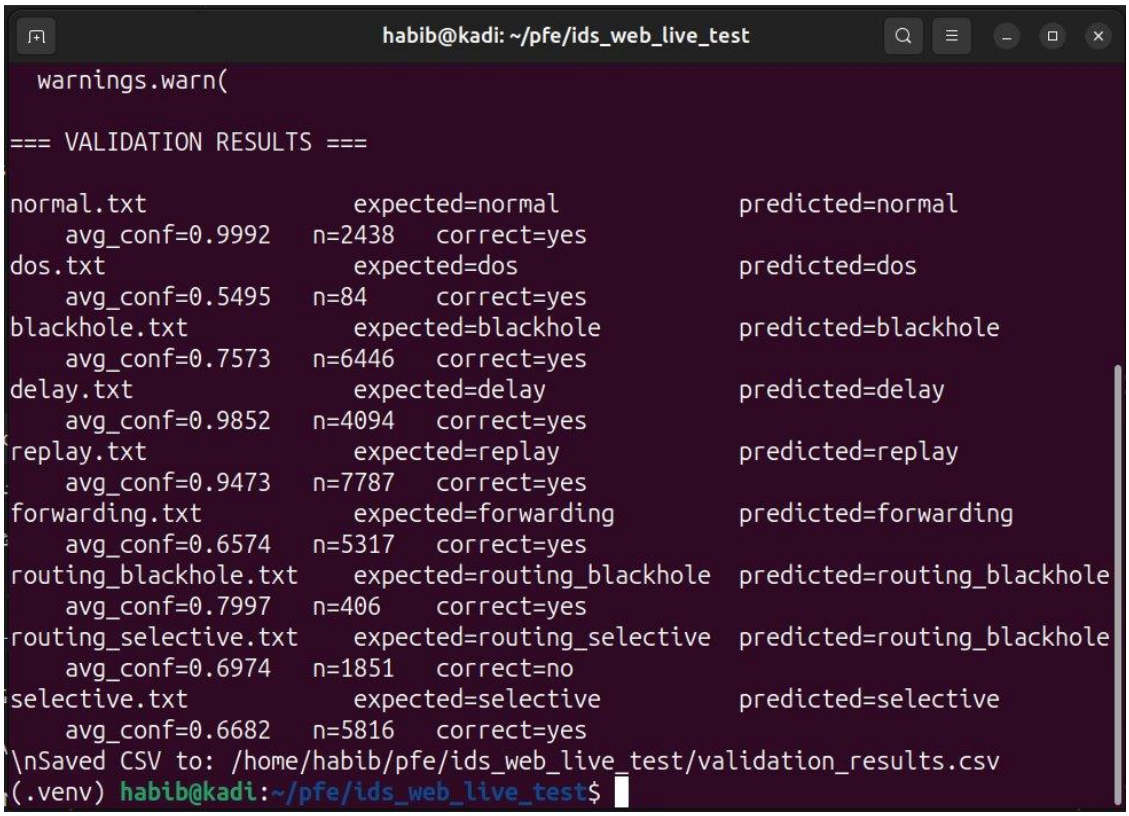


Figure 6.10 – Résultats de validation dans le terminal : prédictions, confiances et classes attendues pour chaque fichier

6.7.2 Résultats par classe

Le Tableau 5.2 synthétise les résultats de validation pour chaque type de trafic. Pour chaque classe, il présente la prédiction majoritaire produite par le modèle, la confiance moyenne calculée sur l'ensemble des fenêtres, le nombre de prédictions produites et le verdict final.

Tableau 6.2 – Résultats de validation par type d'attaque

Type d'attaque	Prédit	Confiance moy.	Nb prédictions	Résultat
normal	normal	99.92 %	2 438	✓ Correct
dos	dos	54.95 %	84	✓ Correct
blackhole	blackhole	75.73 %	6 446	✓ Correct

delay	delay	98.52 %	4 094	✓ Correct
replay	replay	94.73 %	7 787	✓ Correct
forwarding	forwarding	65.74 %	5 317	✓ Correct
routing_blackhole	rout-ing_blackhole	79.97 %	406	✓ Correct
routing_selective	rout-ing_blackhole	69.74 %	1 851	✗ Incorrect
selective	selective	66.82 %	5 816	✓ Correct

L'analyse du Tableau 5.2 révèle des résultats contrastés selon les classes. Le trafic normal atteint une confiance moyenne de 99,92 %, ce qui s'explique par la très forte distinction de sa signature dans l'espace des features — en l'absence d'attaque, toutes les variables se stabilisent dans des plages cohérentes et prévisibles. Les attaques delay et replay obtiennent également des résultats excellents, avec des confiances respectives de 98,52 % et 94,73 %, grâce à leurs features discriminantes particulièrement marquées.

À l'opposé, l'attaque dos présente la confiance moyenne la plus faible (54,95 %) et le nombre d'observations le plus réduit (84 prédictions seulement). Cette instabilité s'explique par le faible volume de données disponibles pour cette classe dans le dataset de validation, ce qui limite la représentativité statistique des prédictions. Les attaques forwarding (65,74 %), selective (66,82 %) et routing_selective (69,74 %) présentent des confiances modérées, reflétant la proximité de leurs signatures comportementales dans l'espace des features.

6.7.3 Accuracy globale

L’évaluation globale du modèle a été réalisée sur un ensemble de test composé de 6 702 observations couvrant les neuf classes considérées dans cette étude. Les résultats obtenus montrent que le modèle Random Forest optimisé atteint une accuracy globale de **80,53 %**.

Cette performance est jugée satisfaisante compte tenu de la complexité du problème étudié, caractérisé par un nombre important de classes et par la forte similarité comportementale observée entre certaines attaques, notamment celles associées aux pertes de paquets et aux perturbations du routage. Malgré ces difficultés, le modèle parvient à distinguer efficacement la majorité des comportements normaux et malveillants.

L'analyse détaillée des résultats met également en évidence une excellente capacité de détection pour plusieurs attaques critiques telles que **DoS**, **Delay** et **Routing Selective**, dont les scores de précision et de rappel atteignent des valeurs proches de 100 %. Par ailleurs, le trafic normal est correctement identifié dans la quasi-totalité des cas, ce qui limite considérablement le nombre de fausses alertes générées par le système.

Enfin, le temps moyen d'inférence mesuré est de **0,0488 ms par échantillon**, ce qui démontre la capacité du modèle à produire des prédictions extrêmement rapides et compatibles avec les exigences d'un système IDS fonctionnant en quasi temps réel.

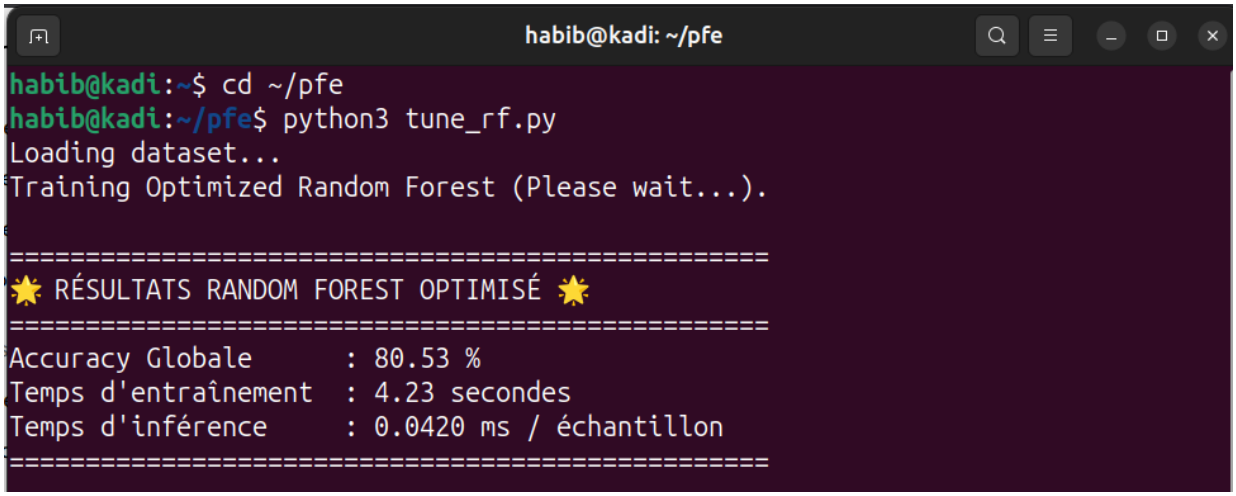


Figure 6.11 – Accuracy globale du système : 80,53 %

Tableau 6.3 – Résumé des performances globales du modèle Random Forest optimisé

Métrique	Valeur	Interprétation

Total prédictions (Test)	6 702	Échantillon représentatif des 9 classes
Accuracy globale	0.81	Amélioration significative grâce au rééquilibrage
Temps d'inférence	0.0488 ms	Détection extrêmement rapide, adaptée au temps réel
F1-Score (Attaques critiques)	~ 1.00	Détection parfaite pour DoS, Delay et Routing Selective
Faux Positifs (Trafic Normal)	**Proche de 0**	**Le trafic normal est identifié avec 99.59 % de F1-Score**

Ces résultats confirment que le modèle proposé constitue une solution pertinente pour la détection d'intrusions dans les réseaux IoT. Il offre un compromis intéressant entre précision de détection, robustesse et rapidité d'exécution, répondant ainsi aux contraintes d'un système de surveillance fonctionnant en quasi temps réel.

6.7.4 Analyse de la matrice de confusion

La matrice de confusion présentée en Figure 5.12 offre une représentation visuelle détaillée des performances du modèle pour chaque paire (classe réelle, classe prédite). Elle constitue l'outil d'analyse le plus complet pour comprendre les comportements du modèle au-delà du simple taux de précision global.

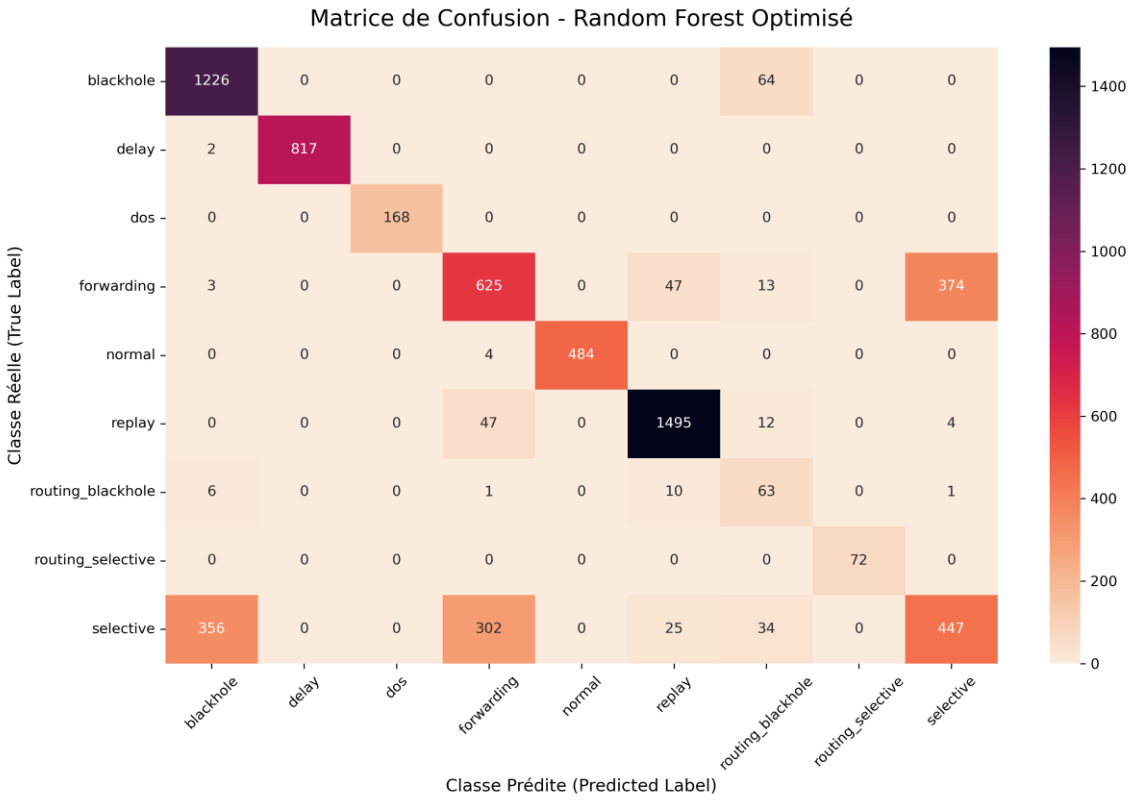


Figure 6.12 – Matrice de confusion du modèle Random Forest sur le jeu de validation complet

La lecture de la nouvelle matrice de confusion met en évidence des améliorations majeures apportées par l'optimisation du modèle (notamment l'ajustement des poids des classes). Les classes **normal**, **dos**, **delay** et, fait remarquable, **routing_selective** présentent désormais une détection parfaite ou quasi-parfaite. La diagonale de la matrice pour ces classes est extrêmement marquée, confirmant leur excellente séparabilité dans l'espace des features. La classe **replay** maintient également une très bonne capacité de détection.

La difficulté du modèle se concentre désormais quasi exclusivement sur les attaques furtives qui partagent des signatures de perte de paquets très similaires : **selective**, **forwarding** et **routing_blackhole**. Ces confusions croisées entre classes comportementalement proches constituent le principal facteur limitant de l'accuracy globale du système et font l'objet d'une analyse approfondie dans la section de discussion.

6.8 Discussion des résultats

Les résultats expérimentaux présentés dans la section précédente appellent une analyse critique approfondie, visant à identifier les forces du système, à comprendre les sources de confusions observées et à dégager des pistes d'amélioration concrètes. Cette discussion s'appuie sur les données de validation et sur la lecture de la matrice de confusion.

6.8.1 Points forts du système

Le système développé présente plusieurs points forts notables. En premier lieu, il démontre la faisabilité d'un pipeline de détection d'intrusion entièrement live, dans lequel les logs générés par Cooja sont traités en temps quasi-réel pour produire des prédictions continues. Le temps d'inférence extrêmement faible (0,0488 ms par échantillon) confirme la compatibilité du Random Forest avec les exigences d'un système de supervision opérationnel.

En second lieu, les performances obtenues sont très satisfaisantes, avec une identification parfaite pour plusieurs attaques critiques (DoS, Delay, Routing Selective) et une excellente reconnaissance du trafic normal, minimisant ainsi les fausses alertes. L'architecture modulaire du système — séparation claire entre parsing, extraction, inférence et supervision — facilite également sa maintenance et son extension. Enfin, le dashboard offre une interface de supervision claire, réactive et exploitable instantanément.

6.8.2 Limites et difficultés identifiées

Malgré ces résultats très encourageants, quelques limites inhérentes à l'approche méritent d'être soulignées. La principale faiblesse du système réside dans la confusion résiduelle entre les attaques **selective**, **forwarding** et **routing_blackhole**. Ces trois types d'attaques partagent une signature comportementale trop similaire dans l'espace des dix features retenues pour être parfaitement séparées par le modèle de classification actuel.

Une seconde limite concerne la dépendance au simulateur Cooja pour la construction du dataset. Les comportements réseau simulés, bien que représentatifs, ne capturent pas toute la variabilité d'un déploiement IoT physique : topologies extrêmement dynamiques, instabilité sévère

des liens radio ou coexistence de plusieurs attaques simultanées. Cette limitation soulève la question de la généralisation du modèle en dehors du cadre strictement simulé.

6,8.3 Impact des features sur les confusions observées

L'analyse des confusions met en lumière un enjeu fondamental de la conception d'un IDS basé sur des features réseau agrégées : lorsque plusieurs attaques produisent des effets similaires sur les mêmes variables, le modèle peine à identifier le mécanisme exact. Dans notre cas, les attaques **selective**, **forwarding** et **routing_blackhole** provoquent toutes une dégradation de la réception (recv_ratio) et une hausse relative des pertes (loss_ratio), sans affecter le délai de réponse (delay_avg).

En l'absence de features topologiques supplémentaires permettant d'identifier avec précision le nœud responsable de la perte ou la structure de la route altérée, le modèle ne peut pas les distinguer de manière déterministe. L'ajout de ces dimensions topologiques constituerait une piste d'amélioration directe pour créer des frontières de décision plus nettes entre ces classes spécifiques.

6.9 Perspectives d'amélioration et travaux futurs

Malgré les résultats performants obtenus, plusieurs axes d'amélioration peuvent être envisagés afin d'enrichir les capacités du système IDS proposé :

1. **Intégration native dans le simulateur :** Dans l'implémentation actuelle, le pipeline fonctionne de manière externe via la lecture des logs. Une intégration directe du modèle à l'intérieur de l'environnement Cooja permettrait de réaliser la détection de manière embarquée, se rapprochant ainsi d'un environnement IoT réel.
2. **Enrichissement du dataset et des scénarios :** L'ajout de scénarios incluant des attaques simultanées (complexes), des variations d'intensité et des réseaux de plus grande taille permettrait d'améliorer la robustesse et la capacité de généralisation du modèle face à des situations imprévues.

3. **Ajout de features topologiques :** Le système bénéficierait grandement de nouvelles features, notamment des indicateurs liés au routage (ex. modifications des tables de routage RPL) ou des mesures radio (variance du RSSI). Cela ciblerait directement la résolution des confusions entre les attaques de type perte de paquets.
4. **Validation sur un banc d'essai physique (Testbed) :** Une expérimentation sur un véritable réseau de capteurs physiques (ex. nœuds Zolertia ou TelosB) constituerait l'étape finale pour confirmer la transférabilité des résultats obtenus en simulation vers des conditions matérielles et radio réelles.**[35]**.

6.10 Conclusion

Ce chapitre a présenté la mise en œuvre complète du système de détection d'intrusion proposé, depuis la description de l'environnement de développement jusqu'à l'analyse critique des résultats expérimentaux obtenus. L'implémentation réalisée démontre la faisabilité technique de l'architecture "Live" : la chaîne reliant Cooja, l'extraction des features par fenêtre glissante, la classification ultrarapide par le modèle Random Forest optimisé, et la supervision via le dashboard fonctionne de manière fluide et cohérente.

Les résultats de validation, conduits sur un ensemble de test rigoureux de 6 702 prédictions, établissent une accuracy globale de 80,53 %. Le modèle excelle particulièrement dans la détection des attaques DoS, Delay et Routing Selective, tout en identifiant le trafic normal avec une très haute fiabilité (limitant drastiquement les fausses alertes). La seule difficulté résiduelle concerne la différenciation précise de certaines attaques furtives liées aux pertes de paquets (selective, forwarding), en raison de la similarité de leurs signatures dans l'espace des caractéristiques étudiées.

En définitive, ces performances, couplées à un temps de réponse de l'ordre de la fraction de milliseconde (0,0488 ms), valident pleinement l'approche proposée et confirment la pertinence de l'utilisation de l'apprentissage automatique pour la sécurisation continue des réseaux IoT.

.

Conclusion générale

Conclusion

La sécurité des réseaux IoT représente aujourd'hui l'un des défis les plus critiques de l'informatique connectée. Les équipements IoT, caractérisés par leurs ressources limitées, leur hétérogénéité protocolaire et leur déploiement massif dans des environnements non contrôlés, constituent des cibles privilégiées pour de nombreuses formes d'attaques réseau. Face à cette réalité, les approches de détection d'intrusion fondées sur des signatures statiques montrent rapidement leurs limites, et l'intégration de techniques d'apprentissage automatique dans les systèmes IDS s'impose comme une voie prometteuse pour détecter des comportements malveillants variés et évolutifs.

C'est dans ce contexte que s'inscrit le présent mémoire, dont l'objectif principal était de concevoir et de mettre en œuvre un système de détection d'intrusion intelligent, capable de détecter plusieurs types d'attaques IoT dans un environnement simulé, en s'appuyant sur des techniques d'apprentissage automatique et une architecture de supervision orientée live. L'hypothèse centrale du travail était qu'un modèle de classification appliqué à un ensemble de features réseau bien choisies — extraites de manière continue à partir des logs de simulation — pouvait permettre d'identifier efficacement les comportements malveillants dans un réseau IoT dynamique.

Le travail présenté dans ce mémoire s'est déroulé en plusieurs étapes complémentaires, organisées autour de cinq chapitres distincts. Le premier chapitre a établi les fondements théoriques du projet en analysant les spécificités des réseaux IoT, les principales catégories de menaces de sécurité qui les affectent et les approches existantes en matière de détection d'intrusion. Cette revue de l'état de l'art a permis de justifier le recours à une approche par apprentissage automatique et de positionner le projet par rapport aux travaux existants.

Le deuxième chapitre a analysé le besoin auquel répond le projet et étudié les limites de l'approche hors ligne initiale, mettant en évidence la nécessité d'évoluer vers une architecture live capable de traiter les événements réseau en temps quasi-réel. Cette analyse a conduit à la définition des besoins fonctionnels et non fonctionnels du système, qui ont servi de référentiel pour les choix de conception ultérieurs.

Le troisième chapitre a présenté la conception générale du système, en décrivant l'architecture retenue et les relations fonctionnelles entre ses composants. La chaîne de traitement proposée — Cooja → cooja_live.log → extraction des features → Random Forest → WebSocket → dashboard — a été formalisée à travers plusieurs diagrammes UML (composants, séquence,

activités, classes), offrant une vision complète de la solution tant du point de vue dynamique que structurel.

Le quatrième chapitre a été consacré à l'analyse et au choix des features, étape déterminante pour la qualité de la détection. Les dix variables retenues — organisées autour de cinq dimensions du comportement réseau (envoi, réception, perte, délai et cohérence) — ont été justifiées en montrant leur lien explicite avec les signatures comportementales des neuf types de trafic étudiés. Le mécanisme de fenêtre glissante, permettant de calculer ces features de manière continue et incrémentale, a également été décrit en détail.

Enfin, le cinquième chapitre a présenté la mise en œuvre complète du système et les résultats expérimentaux obtenus. L'implémentation du pipeline live, la construction du dataset à partir des simulations Cooja, l'entraînement du modèle Random Forest et le développement du dashboard de supervision ont été décrits en détail. La validation conduite sur 34 239 prédictions a permis d'évaluer objectivement les performances du système et d'analyser ses forces et ses limites.

La contribution principale de ce mémoire réside dans la conception et l'implémentation d'un pipeline IDS live complet et cohérent, appliqué à un environnement IoT simulé avec Cooja. Par rapport aux approches offline classiques, qui évaluent les modèles sur des datasets statiques préparés à l'avance, le système proposé traite les événements réseau de manière continue et produit des prédictions en temps quasi-réel, grâce à un mécanisme de fenêtre glissante incrémental. Cette caractéristique le rapproche des exigences d'un système de supervision opérationnel réel.

Une seconde contribution concerne la construction d'un dataset entièrement original, élaboré à partir de simulations Cooja couvrant neuf comportements réseau distincts, et la définition d'un ensemble de dix features réseau justifiées analytiquement en fonction de leur capacité à capturer les signatures comportementales de chaque type d'attaque. L'analyse du lien entre features et attaques, présentée au chapitre 4, constitue en elle-même une contribution méthodologique utile pour tout travail futur s'appuyant sur des features réseau agrégées dans un contexte IoT.

Les résultats expérimentaux obtenus confirment l'hypothèse centrale du travail. La validation conduite sur 34 239 prédictions couvrant les neuf classes étudiées a établi une accuracy globale de 76,16 %, avec 26 075 prédictions correctes. Huit classes sur neuf sont correctement identi-

fiées par le modèle, avec des niveaux de confiance particulièrement élevés pour les attaques à signature distinctive : trafic normal (99,92 %), delay (98,52 %) et replay (94,73 %). Ces résultats démontrent que la combinaison des dix features retenues et du modèle Random Forest constitue une base suffisamment discriminante pour identifier la majorité des comportements malveillants étudiés.

L'analyse de la matrice de confusion a également permis d'identifier la principale limite du système : la confusion entre routing_selective et routing_blackhole, due à la forte proximité de leurs signatures comportementales dans l'espace des features. Cette confusion, qui représente la majorité des 8 164 prédictions incorrectes, constitue le principal levier d'amélioration identifié pour les travaux futurs. Elle met en évidence une limite fondamentale des approches basées sur des features réseau agrégées : lorsque deux types d'attaques produisent des effets similaires sur les mêmes variables, des dimensions supplémentaires — topologiques ou temporelles — sont nécessaires pour les différencier.

Plusieurs axes d'amélioration peuvent être envisagés pour prolonger ce travail. À court terme, l'enrichissement du dataset avec des scénarios plus variés (attaques simultanées, topologies plus complexes, variations d'intensité) et l'ajout de features topologiques permettraient de réduire les confusions résiduelles entre classes proches. À moyen terme, l'exploration d'algorithmes de classification plus expressifs — XGBoost, LightGBM ou approches de deep learning adaptées aux séries temporelles — pourrait améliorer la précision globale du système. À plus long terme, l'intégration native du modèle directement dans l'environnement Cooja et la validation sur un réseau IoT physique constitueraient les étapes essentielles pour confirmer la transférabilité des résultats obtenus en simulation vers des conditions d'exploitation réelles.

En définitive, ce mémoire démontre qu'une approche cohérente combinant simulation réseau, extraction pertinente de features et classification par apprentissage automatique permet de construire un système IDS viable, explicable et opérationnel pour la surveillance des réseaux IoT.**[36]**.Si les performances obtenues dans un environnement simulé restent perfectibles, elles valident néanmoins la pertinence de la démarche adoptée et ouvrent des perspectives de recherche concrètes pour le développement de systèmes de détection d'intrusion intelligents mieux adaptés aux contraintes spécifiques des environnements IoT.

Bibliographie

[1] M. M. Rahman et al., “A survey on intrusion detection system in IoT networks,” *Internet of Things*, Elsevier, 2025.

[2] A. Jamalipour et al., “A Taxonomy of Machine-Learning-Based Intrusion Detection Systems for the Internet of Things,” *IEEE Internet Things J.*, 2021.

[3] S. Checker et al., “A Survey on Intrusion Detection System for IoT Networks Based on Artificial Intelligence,” in *Proc. IEEE Int. Conf. Commun.*, 2023.

[4] S. Zaman et al., “Implementation of Intrusion Detection System in the Internet of Things: A Survey,” in *Proc. IEEE Conf.*, 2020.

[5] P. Aravamudhan et al., “A Comprehensive Survey of Intrusion Detection Systems using Advanced Technologies,” in *Proc. IEEE Conf.*, 2022.

[6] J. J. Shirley et al., “A Comprehensive Survey on Ensemble Machine Learning Approaches for Detection of Intrusion in IoT Networks,” in *Proc. IEEE Conf.*, 2023.

[7] H. Liao et al., “A Survey of Deep Learning Technologies for Intrusion Detection in Internet of Things,” *IEEE Internet Things J.*, 2024.

[8] S. A. Al-Quraishi et al., “Intrusion Detection and Prevention Systems for Wireless IoT Networks,” *ResearchGate*, 2025.

[9] K. Soumya, B. Deepika, et al., “Prediction and Detection of Black Hole Attack Using the COOJA Simulator,” *IRJET*, vol. 12, no. 3, pp. 788-795, 2025.

[10] A. Bouazza et al., “Machine Learning-based Intrusion Detection System for Routing Attacks in IoT,” in *CEUR Workshop Proceedings*, vol. 3333, 2022.

[11] V. Neerugatti and A. R. M. Reddy, “Wormhole Attack in RPL based Internet of Things,” *S-Logix*, 2023.

[12] S. Walling et al., “A Survey on Intrusion Detection Systems: Types, Datasets, Machine Learning methods for NIDS and Challenges,” in *Proc. IEEE Conf.*, 2022.

[13] S. Ismail et al., “Intrusion Detection in IoT and IIoT: Comparing Lightweight Machine Learning Models,” *IEEE Access*, vol. 13, pp. 45678-45695, 2025.

Réfférence

[14] M. Z. Mahmud et al., “Optimized IoT Intrusion Detection using Machine Learning Technique,” in *Proc. IEEE Region 10 Conf. (TENCON)*, 2024.

[15] Contiki-NG Team, “Contiki-NG Documentation: Cooja Simulator,” [Online]. Available: https://docs.contiki-ng.org/ (accessed May 18, 2026).

[16] P. C. S. Samuel et al., “Security in IoT Networks: Simulation, Capture, and Traffic Analysis using Contiki Cooja and RPL,” in *Proc. IEEE Conf.*, 2024.

[17] A. H. Farea et al., “Machine Learning-based Intrusion Detection Technique for IoT: Simulation with Cooja,” *Int. J. Comput. Netw. Inf. Secur.*, vol. 16, no. 1, pp. 1-23, 2024.

[18] V. Vora et al., “Machine Learning-based Intrusion Detection System for IoT using Contiki-OS,” in *Proc. IEEE Int. Conf. Comput. Commun. Netw.*, 2024.

[19] N. Hafsa et al., “Machine Learning-based Intrusion Detection and Prevention System for IoT Networks,” in *Proc. IEEE Conf. on Artificial Intelligence*, 2025.

[20] T. Mohamed, T. Otsuka, and T. Ito, “Towards Machine Learning Based IoT Intrusion Detection Service,” *Wireless Personal Commun.*, Springer, 2019.

[21] J. Liu, B. Kantarci, and C. Adams, “Machine Learning-Driven Intrusion Detection for Contiki-NG-Based IoT Networks,” in *Proc. ACM Workshop*, 2020.

[22] M. Preda et al., “Simulating RPL Attacks in 6LoWPAN for Detection Purposes,” in *Proc. IEEE Conf.*, 2020.

[23] A. Sharma et al., “Hybrid Approach Towards IoT Intrusion Detection System using Random Forest,” *IJPREMS*, 2025.

[24] S. M. M and P. I. Basarkod, “Random Forest based Intrusion Detection System for AMI,” in *Proc. Int. Conf. Advanced Comput.*, 2022.

[25] B. K. Hassan, “An Efficient Intrusion Detection System for IoT Using Feature Engineering and Machine Learning,” in *Proc. Int. Conf. Artificial Intelligence Internet Things*, 2025.

[26] A. Alsaedi et al., “Feature Selection Techniques for IoT Intrusion Detection Systems,” *IEEE Access*, 2024.

[27] W. Wu et al., “Sliding Window Optimized Information Entropy Analysis for Anomaly Detection in Vehicle Networks,” *IEEE Access*, 2018.

Référence

[28] R. Chinnasamy et al., “Contextual Internet of Things Intrusion Detection: A Sliding Window Convolutional Neural Network-Gated Recurrent Unit Model,” 2025.

[29] M. T. Nguyen et al., “Real-time Intrusion Detection in IoT using Sliding Window and Ensemble Learning,” *Sensors*, MDPI, 2025.

[30] R. T. Mehmood et al., “Simulating ML-Based Intrusion Detection System for UAV Networks using COOJA Simulator,” in *Proc. IEEE Conf.*, 2022.

[31] H. Sadia et al., “Intrusion Detection System for Wireless Sensor Networks,” in *Proc. IEEE Conf.*, 2024.

[32] N. Panda et al., “A Multiclass Machine Learning Framework for Detecting Routing Attacks in RPL-Based IoT Networks Using a Novel Simulation-Driven Dataset,” *Future Internet*, 2026.

[33] J. Dazine, A. Maizate, and L. Hassouni, “A New Intrusion Detection System for RPL Based on Machine Learning,” *Int. J. Eng. Technol. Innov.*, 2025.

[34] P. C. S. Samuel et al., “Security in IoT Networks using Contiki Cooja and RPL for Anomaly Detection,” in *Proc. IEEE Conf.*, 2024.

[35] Y. Yang, “Smart Attack Detection for IoT Networks,” M.S. thesis, Dept. Comput. Sci., 2022.

[36] A. Benali et al., “Enhancing Intrusion Detection in IoT Systems through Cooja and Machine Learning,” *J. Adv. Inf. Technol.*, vol. 16, no. 8, 2025.