



TÉLÉCOM PHYSIQUE STRASBOURG

RAPPORT DE PROTECTION DES DONNÉES

Analyse de données cyberphysiques sur la distribution d'eau

Nathan CERISARA

Clément DESBERG

Ysée JACQUET

Lucas LEVY

21 janvier 2026

Table des matières

Introduction	1
1 Données	1
1.1 Visualisation et analyse des données	2
1.1.1 Présentation générale des données	2
1.1.2 Présentation parallèle des données réseau et physiques	3
1.1.3 Valeurs manquantes	5
1.2 Pré-traitement	6
2 Application des algorithmes	7
2.1 Algorithmes utilisés	7
2.2 Organisation des entraînements	7
2.3 Hyperparamètres et résultats	8
3 Évaluation et comparaison avec les données de référence	9
3.1 Indicateurs et résultats principaux	9
3.1.1 Pour le jeu de données réseau	9
3.1.2 Pour le jeu de données physiques	10
3.2 Analyse des erreurs	12
Conclusion	14
Références	15

Table des figures

1	Nombre d'entrées catégorisées normal et attack dans les jeux de données .	2
2	Nombre d'entrées par fichier de données réseau d'après [2]	3
3	Nombre d'entrées par fichier de données physiques d'après [2]	3
4	Répartition des types d'attaques dans les données réseau	4
5	Répartition des types d'attaques dans les données physiques	4
6	Pourcentage de valeurs manquantes par colonne dans les données réseau . .	6
7	Matrice de corrélation des erreurs entre les modèles pour les données réseau	12
8	Matrice de corrélation des erreurs entre les modèles pour les données physiques	13

Liste des tableaux

1	Aperçu des fichiers de données fournis, d'après le fichier fourni README.xlsx	1
2	Répartition des données par label	2
3	Caractéristiques des données réseau	5
4	Caractéristiques des données physiques	5
5	Comparaison des expériences sur le jeu de données network	9
6	Résultats de l'évaluation des algorithmes d'apprentissage automatique sur le jeu de données physique	10
7	Comparaison des expériences sur le jeu de données physiques	11
8	Résultats de l'évaluation des algorithmes d'apprentissage automatique sur le jeu de données physique	11
9	Échantillons physiques systématiquement mal classés lors des différentes exécutions	14
10	20 échantillons réseau systématiquement mal classés lors des différentes exécutions	14

Abréviations employées

- **CART** : Classification And Regression Trees
- **CNN** : Convolutional Neural Network
- **CSV** : Comma Separated Values
- **FT Transformer** : Feature Tokenizer Transformer
- **KNN** : K-Nearest Neighbors
- **MITM** : Man-In-The-Middle
- **MLP** : Multi-Layer Perceptron

- **MCC** : Matthews Correlation Coefficient
- **NLP** : Natural Language Processing

Introduction

Ce projet consiste en l'analyse de données cyberphysiques issues d'un système de distribution d'eau potable [1]. L'objectif principal est de développer et d'évaluer des modèles de machine learning capables de détecter et de classer les anomalies dans le système, telles que les attaques informatiques ou les défaillances physiques. Les consignes du projet prévoient l'implémentation de K-Nearest Neighbors (KNN), Random Forest, XGBoost et Multi-Layer Perceptron (MLP) et Classification And Regression Trees (CART). Il est également possible de remplacer l'un de ces modèles par un ou plusieurs autres au choix, ce qui a été fait en implémentant des modèles basés sur des transformeurs (Tab Transformer et FT Transformer) ainsi qu'un MLP avec attention à la place du modèle CART. Finalement, les performances des modèles développés ont été comparées avec celles fournies dans l'article de référence [2].

1 Données

L'ensemble des données comprend 5 fichiers Comma Separated Values (CSV) sur les relevés physiques et 5 fichiers CSV sur les relevés réseau. Dans chaque cas, il y a un fichier de mesures prises en période normale – sans anomalie – et 4 fichiers de mesures prises en période anormale, avec des anomalies correspondant au type de données (attaques informatiques ou défaillances physiques). Le sujet de chaque fichier est résumé dans la [Table 1](#) ci-dessous :

Fichier	Contenu
normal.csv	Acquisitions du trafic réseau en période normale
phy_norm.csv	Acquisitions des mesures physiques en période normale
attack_1.csv	Première acquisition du trafic réseau avec anomalies
dataset_att_1.csv	Première acquisition des mesures physiques avec anomalies
attack_2.csv	Deuxième acquisition du trafic réseau avec anomalies
dataset_att_2.csv	Deuxième acquisition des mesures physiques avec anomalies
attack_3.csv	Troisième acquisition du trafic réseau avec anomalies
dataset_att_3.csv	Troisième acquisition des mesures physiques avec anomalies
attack_4.csv	Quatrième acquisition du trafic réseau avec anomalies
dataset_att_4.csv	Quatrième acquisition des mesures physiques avec anomalies

TABLE 1 – Aperçu des fichiers de données fournis, d'après le fichier fourni `README.xlsx`

Il y a également des fichiers packet capture (PCAP) fournis pour les données réseau, mais ceux-ci n'ont pas été utilisés dans le cadre de ce projet.

1.1 Visualisation et analyse des données

1.1.1 Présentation générale des données

Dans chaque fichier CSV, une ligne représente un ensemble de données collectées à un instant donné, tandis que les colonnes renvoient à différentes informations relevées ou différents dispositifs sur lesquels une mesure a été faite. Chaque ligne est associée à une étiquette (`label`) indiquant si les données correspondent à un état normal (étiquette `normal` ou altérée en `nomal`) ou à une anomalie (étiquettes `anomaly`, `DoS`, `MITM`, `scan` ou `physical fault`). La répartition globale peut être observée dans la Figure 1, tandis que le détail de la répartition des étiquettes peut être consulté dans la Table 2.



FIGURE 1 – Nombre d'entrées catégorisées `normal` et `attack` dans les jeux de données

Étiquette	Données totales	Données réseau	Données physiques
normal	20 461 956	20 453 299	8 657
nomal	249	0	249
Total normales	20 462 205	20 453 299	8 906
DoS	5 671 852	5 671 542	310
MITM	2 156 417	2 155 409	1 008
physical fault	1 549 189	1 548 504	685
anomaly	389	389	0
scan	75	61	14
Total attaques	9 377 922	9 375 905	2 017
TOTAL	29 840 127	29 829 204	10 923

TABLE 2 – Répartition des données par label

On constate qu'il y a beaucoup moins de données physiques que réseau (10 923 contre 29 829 204), ce qui est logique étant donné que les mesures physiques sont prises toutes les secondes, tandis que les relevés réseau sont probablement effectués à chaque paquet transmis (donc plusieurs par seconde). En effet, [2] indique que les données réseau ont été capturées à l'aide de *Wireshark*, un outil d'analyse de paquets réseau, ce qui confirme cette hypothèse.

1.1.2 Présentation parallèle des données réseau et physiques

Les statistiques obtenues sur les données réseaux et physiques sont identiques à celle de [2]. Ainsi, on peut observer le nombre d'entrées pour chaque fichier de données dans les Figure 2 et Figure 3.

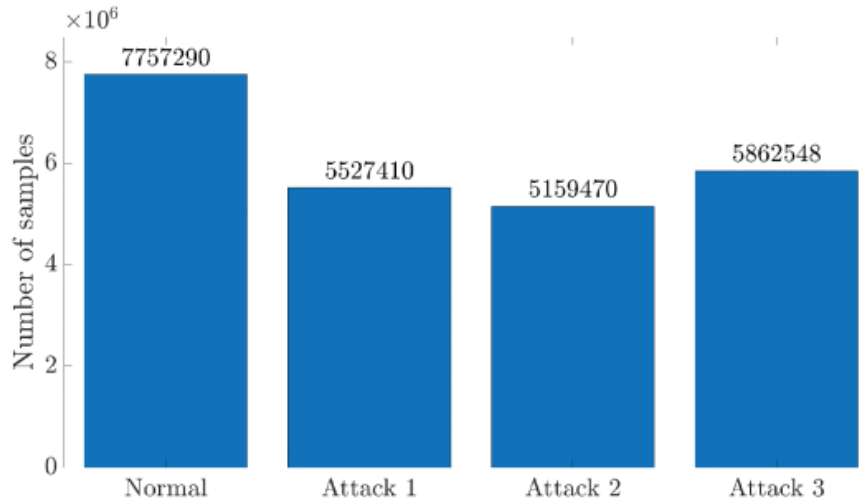


FIGURE 2 – Nombre d'entrées par fichier de données réseau d'après [2]

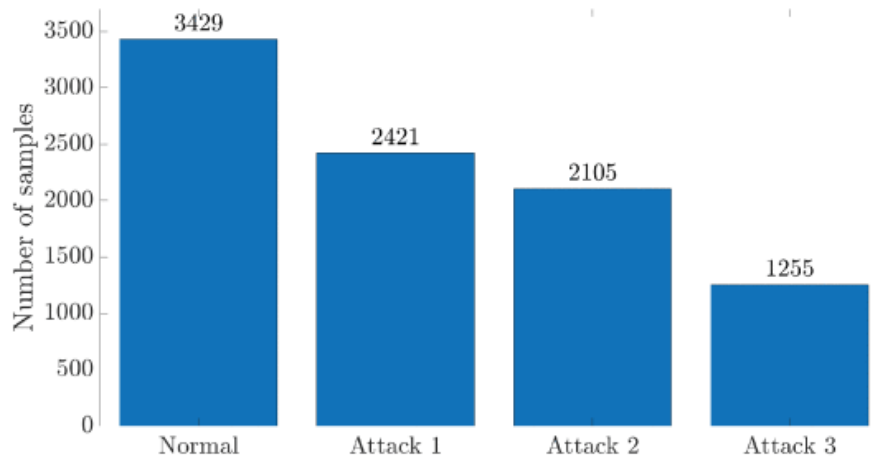


FIGURE 3 – Nombre d'entrées par fichier de données physiques d'après [2]

Il manque cependant les informations sur les fichiers `attack_4.csv` et `dataset_att_4.csv` : on y a relevé respectivement **5 522 490** et **1 717** entrées, ce qui reste dans la même ordre de grandeur que les autres fichiers d'attaques pour chaque type.

Par ailleurs, on constate dans les deux cas que comparé au fichier dit "normal", les fichiers d'attaques sont plus petits lorsque pris indépendamment. Cela est expliqué dans [2] par le fait que les périodes d'attaques sont plus courtes que la périodes normale qui a été relevée :

Specifically, the first acquisition lasts 1 hour and shows a total of 12 process cycles : it refers to the WDT while working in normal conditions without any attack. On the contrary, the remaining three acquisitions, which last 60 minutes, provide 8, 7 and 4 process cycles respectively.

On dispose également de statistiques sur la répartition des différents types d'attaques dans les données, présentées dans les [Figure 4](#) et [Figure 5](#).

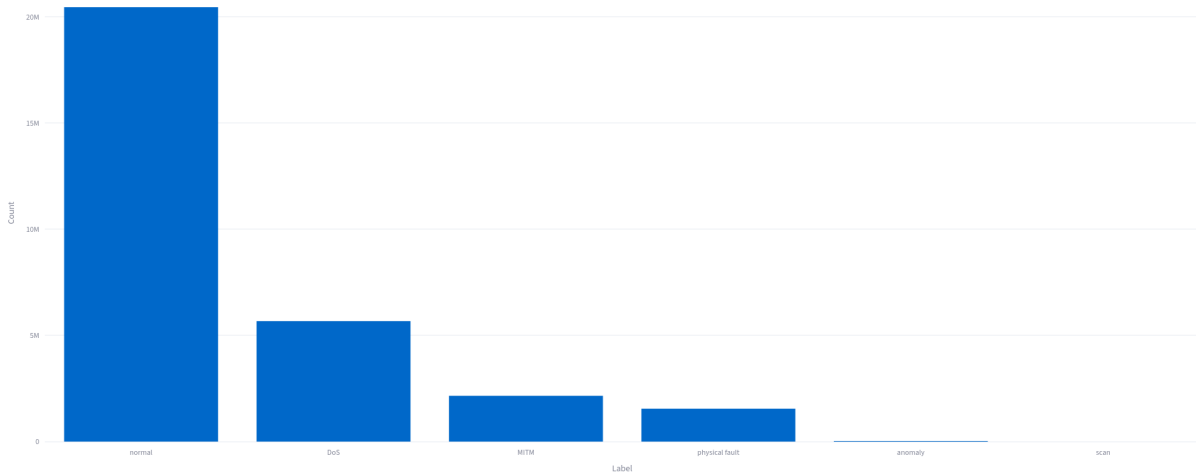


FIGURE 4 – Répartition des types d'attaques dans les données réseau

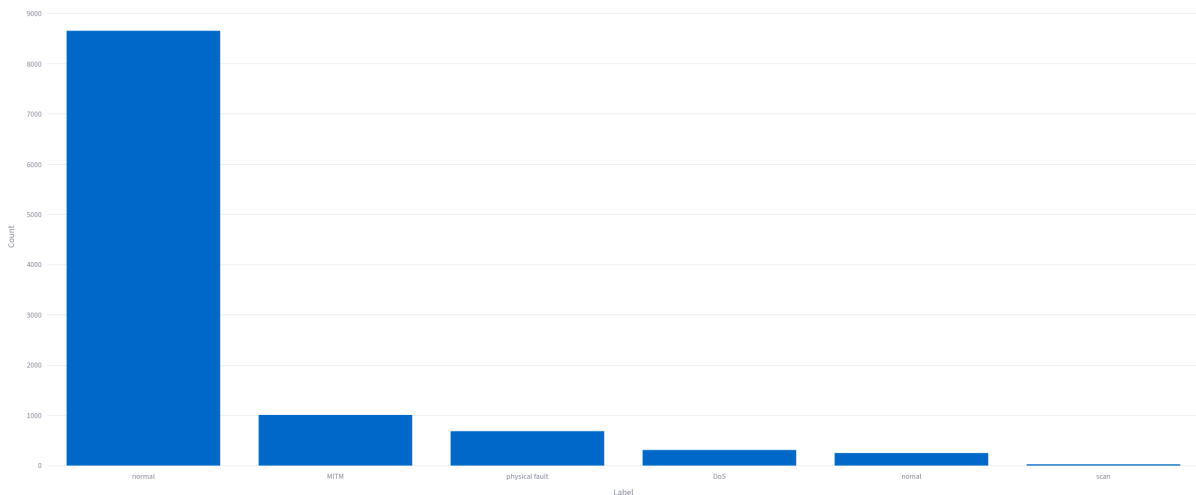


FIGURE 5 – Répartition des types d'attaques dans les données physiques

On voit que certains types d'attaques sont très représentés, notamment DoS et MITM, tandis que d'autres sont très peu présents, comme `scan` dans les deux types de données. Il est donc important de prendre en compte cette répartition lors de l'entraînement des modèles, afin d'éviter qu'ils ne soient biaisés en faveur des classes majoritaires. Il est probable que les classes `scan` et `anomaly` soient également très difficiles à cerner par les

modèles.

Aussi, le contenu des différentes colonnes est détaillé dans [2]. Les données récupérées via *Wireshark* en comportent **14**, listées dans la [Table 3](#) (une traduction du tableau original présent dans [2]).

N°	Nom	Description
1	Time	Date d'acquisition
2	Src IP address	Adresse IP source
3	Dst IP adress	Adresse IP destination
4	Src MAC address	Adresse MAC source
5	Dst MAC address	Adresse MAC destination
6	Src Port	Port source
7	Dst Port	Port destination
8	Proto	Protocole
9	TCP flags	CWR ECN URG ACK PSH RST SYN FIN flags
10	Payload size	Taille de la charge utile
11	MODBUS code	Code de fonction MODBUS
12	MODBUS value	Valeur réponse MODBUS
13	num_pkts_src	Nombre de paquets de la même adresse source durant les 2 dernières secondes
14	num_pkts_dst	Nombre de paquets de la même adresse destination durant les 2 dernières secondes

TABLE 3 – Caractéristiques des données réseau

Les mesures physiques, quant à elles, concernent des paramètres tels que l'état des pompes, les valeurs de différents capteurs, etc. En tout, il y en a **41**, présentées dans la [Table 4](#) (également une traduction du tableau original présent dans [2]).

N°	Nom	Description
1	Time	Date d'acquisition
2-9	Tank_i	Valeur du capteur de pression du réservoir i (i allant de 1 à 8)
10-15	Pump_i	État de la pompe i (i allant de 1 à 6)
16-19	Flow_sensor_i	Valeur du capteur de débit i (i allant de 1 à 4)
20-41	Valv_i	État de l'électrovanne i (i allant de 1 à 22)

TABLE 4 – Caractéristiques des données physiques

1.1.3 Valeurs manquantes

Bien entendu, une vérification a été faite pour détecter la présence de valeurs manquantes dans les données. En l'occurrence, les données physiques ne présentent pas de valeurs manquantes. Concernant les données réseau, elles sont présentées dans la [Figure 6](#) :

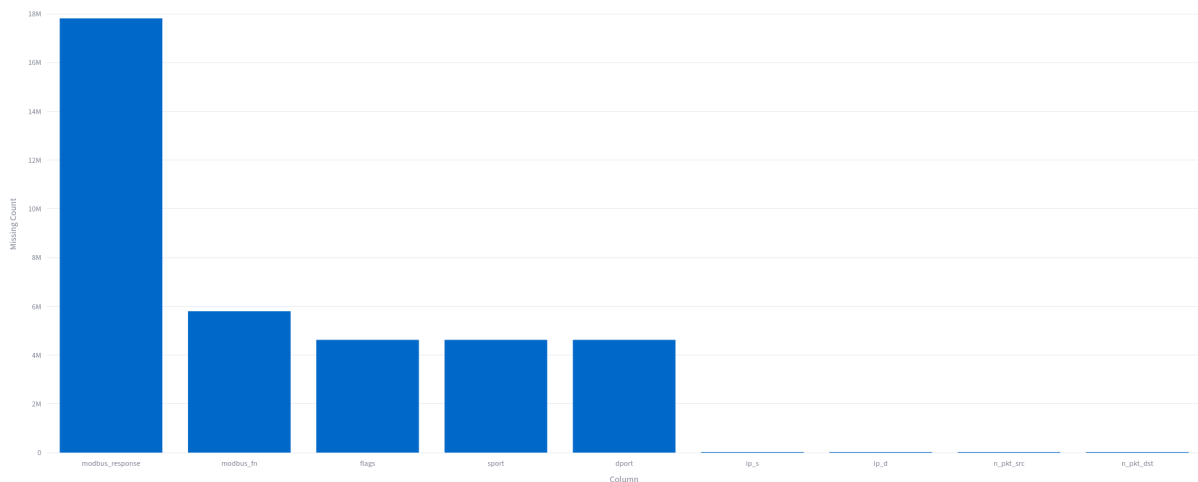


FIGURE 6 – Pourcentage de valeurs manquantes par colonne dans les données réseau

1.2 Pré-traitement

L'ensemble des données est soumis à un pré-traitement général avant d'être utilisé par les différents modèles :

1. Les données brutes (physique et réseau) sont chargées depuis les différents fichiers CSV.
2. Les jeux de données étant particulièrement lourds, ils sont sous-échantillonnés pour permettre de travailler sur des sous-ensembles réduits.
3. Les caractéristiques et les labels sont extraits.
4. Les valeurs manquantes (cf. [1.1.3 Valeurs manquantes](#)) sont remplacés par des 0.
5. Les labels sont encodés avec `LabelEncoder` de `scikit-learn` [3], ce qui permet de convertir les étiquettes textuelles en valeurs numériques.
6. Les données sont stratifiées en ensembles d'entraînement, de validation et de test avec `train_test_split` de `scikit-learn`, en utilisant les options de `stratify` et `shuffle` pour garantir une répartition équilibrée des classes dans chaque ensemble. Le ratio utilisé est de 70% pour l'ensemble d'entraînement, 15% pour l'ensemble de validation et 15% pour l'ensemble de test.
7. Si le *balancing* est activé, il est appliqué sur l'ensemble d'entraînement à l'aide de la fonction `RandomOverSampler` de la bibliothèque `imbalanced-learn` [4].
8. Enfin, les données sont normalisées avec `StandardScaler` de `scikit-learn` pour garantir que chaque caractéristique ait une moyenne nulle et une variance unitaire.

2 Application des algorithmes

2.1 Algorithmes utilisés

Au total, 7 algorithmes différents ont été implémentés pour ce projet. Pour quatre d'entre eux, trois tailles de modèles ont été testées : `small`, `medium` et `large` ; les autres n'ont été testés qu'en taille `small` et `medium`. Cela sera expliqué plus en détail dans la section [2.3 Hyperparamètres et résultats](#).

Parmi les modèles demandés, les suivants ont été implémentés :

- **KNN** (en `small`, `medium` et `large`)
- **Random Forest** (en `small`, `medium` et `large`)
- **XGBoost** (en `small` et `medium`)
- **MLP** (en `small`, `medium` et `large`)

Le modèle CART n'a pas été implémenté, mais a été remplacé par les trois modèles ci-dessous :

- **Tab Transformer** (en `small` et `medium`) : modèle basé sur des transformeurs appliqués aux données tabulaires, utilisant des embeddings pour les variables catégorielles et un mécanisme d'auto-attention pour capturer les interactions entre les caractéristiques.
- **FT Transformer** (en `small` et `medium`) : variante optimisée du transformeur pour données tabulaires, combinant embeddings de caractéristiques et couches d'auto-attention avec une tête de prédiction dédiée, offrant de bonnes performances tout en restant peu coûteuse en calcul.
- **MLP avec attention** (en `small`, en `medium` et `large`) : même architecture que le MLP classique, mais avec un mécanisme d'attention ajouté.

2.2 Organisation des entraînements

L'ensemble du code est organisé pour permettre une expérimentation facile et reproductible des différents modèles sur les deux types de données.

Les données sont pré-traitées de manière cohérente pour tous les modèles (voir [1.2 Pré-traitement](#)), et un système de cache est mis en place pour éviter de répéter le pré-traitement lors de nouvelles exécutions. Il est enregistré dans le dossier `cached_datasets`.

Chaque modèle est défini dans un script séparé contenu dans un dossier `models/`, avec des configurations spécifiques (`configs/model_type/config_name.yaml`) pour chaque type de modèle : hyperparamètres, jeux de données, taille des modèles, etc. Les scripts d'entraînement et de test sont situés dans les fichiers `scripts/train.py` et `scripts/test.py`. Les différents modèles sont entraînés individuellement avec leur script d'entraînement

et leur fichier de configuration, ou bien plusieurs modèles peuvent être entraînés simultanément avec le script `scripts/train_all.py`.

Les résultats de chaque entraînement sont enregistrés de manière structurée dans le dossier `experiments` en fonction de leur configuration, permettant une analyse et une comparaison faciles des performances des modèles. Le détail des résultats enregistrés est présenté dans la section suivante.

2.3 Hyperparamètres et résultats

En termes d'hyperparamètres, il n'y en a qu'un seul qui a été vraiment manipulé : la taille des modèles utilisés. En effet, pour chaque type de modèle implémenté, deux ou trois tailles de modèles ont été testées : `small`, `medium` et parfois `large`. La différence principale entre elles réside dans le nombre de couches et de neurones utilisés et le nombre de données utilisées pour l'apprentissage (il faut *scaler* avec la taille du modèle). Cependant, étant donné la taille réduite du dataset physique, on utilisera toujours toutes ses données alors qu'on *scalera* sur le réseau. Par exemple, pour le MLP, le modèle `small` utilise 2 couches cachées avec 32 et 64 neurones et est entraîné sur 300 000 données réseau, tandis que le modèle `medium` utilise 3 couches cachées de 64, 128 et 256 neurones et est entraîné sur 500 000 données réseau.

Concernant les résultats, sont enregistrés :

- les poids du meilleur modèle,
- la configuration utilisée (au cas où les configurations changent),
- Les métriques sur les jeux de données d'entraînement, validation et test du meilleur modèle,
- Les erreurs sur les jeux de données d'entraînement, validation et test du meilleur modèle,
- Les courbes de loss et d'accuracy sur les jeux de données d'entraînement et de validation.

Deux rapports récapitulatifs de tous les entraînements sont ensuite générés avec le script `src/analyze_results.py`, un pour les jeux de données physiques `results_analysis/physical/` et un pour les jeux de données réseaux `results_analysis/network/report.md`. On y trouve :

- un tableau comparatif des performances de chaque modèle, ordonné par F1-score (macro) décroissant,
- un tableau récapitulatif des classes mal classées par chaque modèle, ordonné par taux d'erreur décroissant,

- un tableau récapitulatif du top 20 des entrées les moins bien classées sur l'ensemble des modèles, avec la prédiction majoritaire,
- une matrice de corrélation des erreurs entre les différents modèles,
- l'ensemble des courbes d'entraînement pour tous les modèles.

3 Évaluation et comparaison avec les données de référence

3.1 Indicateurs et résultats principaux

3.1.1 Pour le jeu de données réseau

L'évaluation sur le jeu de données réseau (Table 5) montre une dispersion des performances beaucoup plus marquée que sur les données physiques. Avec plusieurs centaines de millier d'entrées, la capacité de généralisation des modèles est mise à l'épreuve. On observe que les modèles "small", bien que plus simples, dominent ici le classement. Cela suggère que l'augmentation de la complexité des modèles (tailles "medium" et "large") sans un réglage extrêmement fin des hyperparamètres ou une augmentation proportionnelle du temps d'entraînement mène rapidement à une dégradation des performances sur ce type de trafic.

Rang	Modèle	Précision	F1 (macro)	Précision équilibrée	MCC	Temps (s)
1	small_random_forest	0.8760	0.8688	0.9592	0.5627	43.0
2	small_knn	0.8990	0.8368	0.8442	0.3495	5.3
3	small_ft_transformer	0.8552	0.7743	0.9598	0.5401	2673.9
4	small_mlp	0.8510	0.7718	0.9585	0.5333	745.1
5	large_random_forest	0.7854	0.7490	0.9382	0.6018	744.8
6	large_xgboost	0.7786	0.7456	0.9377	0.5965	364.1
7	small_tab_transformer	0.4656	0.7042	0.8548	0.2300	2786.7
8	medium_xgboost	0.7113	0.6961	0.9279	0.4693	680.7
9	medium_xgboost	0.7113	0.6961	0.9279	0.4693	381.5
10	medium_random_forest	0.7058	0.6843	0.8724	0.4665	641.2
11	medium_random_forest	0.7058	0.6843	0.8724	0.4665	756.0
12	large_knn	0.8708	0.6672	0.6907	0.5412	4.3
13	small_xgboost	0.8663	0.6134	0.7095	0.5515	26.5
14	small_attention_mlp	0.6476	0.5953	0.8121	0.1380	1308.3
15	medium_knn	0.8589	0.5909	0.5706	0.2198	18.4
16	medium_knn	0.8589	0.5909	0.5706	0.2198	17.3
17	medium_attention_mlp	0.7089	0.5452	0.7450	0.4650	161.1
18	large_attention_mlp	0.7700	0.4577	0.6085	0.5831	2889.7
19	medium_mlp	0.6955	0.4523	0.7653	0.4563	24.8
20	medium_ft_transformer	0.7108	0.4457	0.6015	0.4658	385.7
21	medium_tab_transformer	0.8820	0.3215	0.3307	0.1420	268.6
22	large_mlp	0.4899	0.2944	0.4081	0.3221	2994.4

TABLE 5 – Comparaison des expériences sur le jeu de données network

En comparant nos résultats avec ceux de la Table 6, on note une tendance intéressante : nos meilleurs modèles (small_random_forest et small_knn) obtiennent une exac-

titude (accuracy) supérieure à celle rapportée dans l'article (0,87-0,89 contre 0,75-0,77). De même, nos F1-scores macro sont nettement plus élevés (0,86 contre 0,54).

Cette supériorité peut s'expliquer par notre stratégie de pré-traitement incluant le sous-échantillonnage et la normalisation, ou par une meilleure adaptation des hyperparamètres de nos modèles "small". Toutefois, il est important de noter que le temps de calcul des modèles complexes comme le `large_mlp` ou le `ft_transformer` explose sur ce dataset, sans pour autant garantir de meilleurs résultats, ce qui valide l'efficacité des méthodes arborescentes et des voisins les plus proches pour la détection d'intrusions réseau.

Algorithme	Exactitude	Rappel	Précision	F1-score
KNN	0,77	0,44	0,68	0,53
RF	0,75	0,53	0,56	0,54
SVM	0,69	0,99	0,10	0,20
NB	0,75	0,15	0,90	0,27

TABLE 6 – Résultats de l'évaluation des algorithmes d'apprentissage automatique sur le jeu de données physique

3.1.2 Pour le jeu de données physiques

D'après les résultats présentés dans la [Table 5](#), le modèle `large_xgboost` obtient les meilleures performances sur le jeu de données `physical`, avec une précision de **98,60%**, un F1-macro de **0,8179**, une précision équilibrée de **0,8174** et un MCC de **0,9609**. Le temps d'entraînement est également très faible (quelques secondes), ce qui en fait un choix efficace pour ce type de données. À l'inverse, les modèles `small_random_forest` ou `large_ft_transformer` affichent les performances les plus faibles, ce qui peut s'expliquer par un manque de capacité de généralisation pour le premier (sous-apprentissage dû à une structure trop restreinte) et une difficulté de convergence ou un sur-ajustement (overfitting) pour le second, les architectures de type Transformer étant souvent plus complexes à optimiser sur des données tabulaires de taille réduite.

Rang	Modèle	Précision	F1 (macro)	Précision équilibrée	MCC	Temps (s)
1	large_xgboost	0.9860	0.8179	0.8174	0.9609	3.9
2	medium_xgboost	0.9872	0.8134	0.8075	0.9641	3.1
3	large_random_forest	0.9866	0.8132	0.8001	0.9622	0.9
4	medium_tab_transformer	0.9780	0.8129	0.8270	0.9419	249.2
5	medium_knn	0.9793	0.8095	0.8249	0.9450	0.0
6	small_k5_dist_knn	0.9793	0.8095	0.8249	0.9450	0.0
7	medium_knn	0.9793	0.8095	0.8249	0.9450	0.0
8	large_knn	0.9823	0.8053	0.8079	0.9505	0.0
9	small_knn	0.9774	0.8007	0.8235	0.9402	0.0
10	small_tab_transformer	0.9683	0.8001	0.8250	0.9188	40.7
11	medium_random_forest	0.9695	0.7936	0.8150	0.9198	0.6
12	medium_attention_mlp	0.9365	0.7612	0.8163	0.8510	108.1
13	small_k10_manhattan_knn	0.9390	0.7531	0.8164	0.8564	0.0
14	large_tab_transformer	0.9475	0.7352	0.7068	0.8504	66.5
15	small_attention_mlp	0.9048	0.7259	0.8106	0.7946	35.2
16	medium_ft_transformer	0.9042	0.7178	0.7954	0.7806	126.6
17	medium_attention_mlp	0.8969	0.7129	0.8080	0.7809	62.3
18	medium_ft_transformer	0.8847	0.6974	0.7991	0.7601	121.0
19	small_k20_uniform_knn	0.8908	0.6969	0.8062	0.7722	0.0
20	small_mlp	0.8786	0.6889	0.7988	0.7515	22.7
21	medium_mlp	0.8700	0.6864	0.7902	0.7397	49.3
22	small_ft_transformer	0.8401	0.6593	0.7970	0.7004	56.1
23	small_xgboost	0.8676	0.6412	0.7906	0.7287	0.6
24	large_mlp	0.7724	0.5396	0.7633	0.6137	20.9
25	large_attention_mlp	0.6949	0.4734	0.7326	0.5333	13.1
26	small_random_forest	0.6101	0.4687	0.7242	0.4779	0.2
27	large_ft_transformer	0.6992	0.4644	0.6359	0.5176	82.2

TABLE 7 – Comparaison des expériences sur le jeu de données physiques

En comparant les résultats obtenus avec ceux de l'article de référence [2], on constate que nos modèles affichent des performances globales très compétitives en termes d'exactitude (accuracy), bien que le F1-score macro suggère une difficulté persistante à classer parfaitement certaines classes minoritaires par rapport aux résultats de l'article. En effet dans la Table 6, on peut voir que les performances des KNN, Random Forest, SVM et Naive Bayes (NB) sont sensiblement supérieures aux nôtres en termes de F1-score. Cette différence peut provenir d'un prétraitement des données distinct, d'une gestion différente du déséquilibre des classes, ou de l'utilisation d'une moyenne pondérée (weighted) plutôt que macro pour le calcul du F1-score dans l'article de référence.

Algorithme	Exactitude	Rappel	Précision	F1-score
KNN	0,98	0,95	0,95	0,95
RF	0,99	0,98	0,95	0,97
SVM	0,93	0,92	0,64	0,75
NB	0,93	0,92	0,66	0,77

TABLE 8 – Résultats de l'évaluation des algorithmes d'apprentissage automatique sur le jeu de données physique

3.2 Analyse des erreurs

En s'attardant sur les erreurs commises par les différents modèles, on peut observer certaines corrélation entre celles-ci, comme le montre les [Figure 6](#) et [Figure 7](#). Par exemple — et sans surprise — le modèle MLP et le modèle MLP avec attention présentent une forte corrélation dans leurs erreurs.

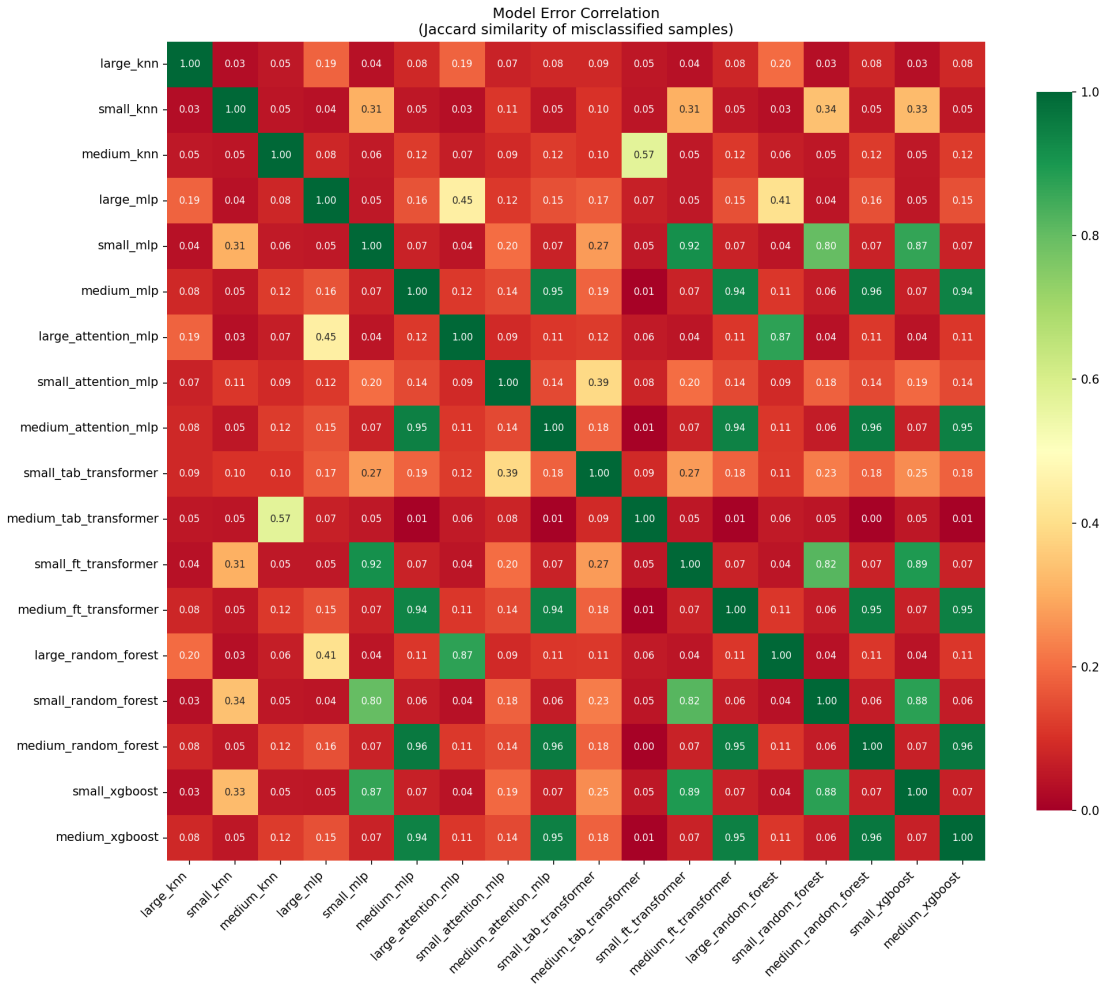


FIGURE 7 – Matrice de corrélation des erreurs entre les modèles pour les données réseau

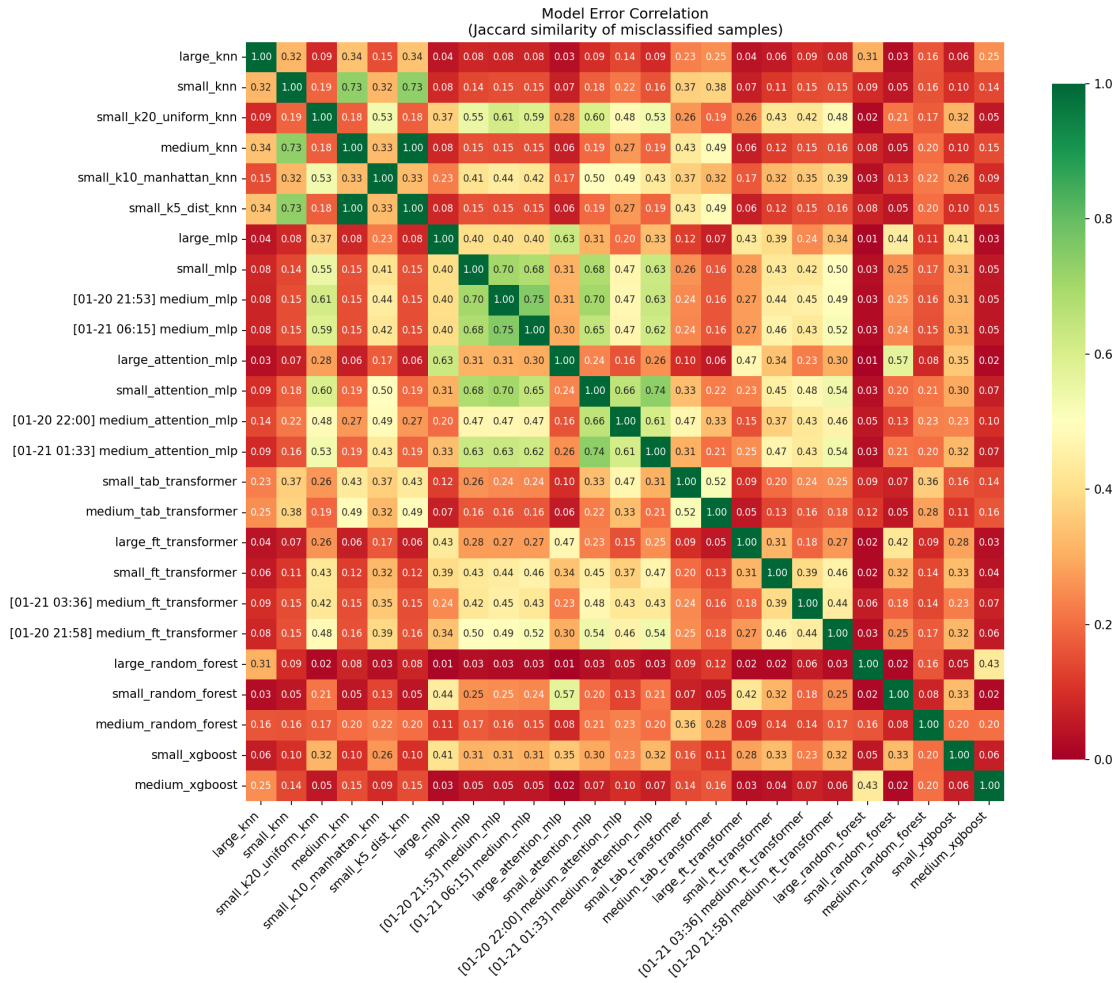


FIGURE 8 – Matrice de corrélation des erreurs entre les modèles pour les données physiques

Comme il y a très peu de faible corrélations, il est raisonnable de penser que certains échantillons sont systématiquement mal classés par tous les modèles. La Table 6 liste les échantillons qui ont été mal classés par l'ensemble des modèles lors de nos différentes exécutions, ainsi que la prédiction majoritaire effectuée par les modèles pour ces échantillons. Heureusement, on constate que leur nombre est très faible — seulement 17 échantillons sur un total de X — donc il n'y a pas beaucoup d'entrées réellement problématiques.

On remarque que la majorité des échantillons mal classés appartiennent à la classe **normal**, ce qui indique que les modèles ont du mal à distinguer les échantillons normaux des anomalies dans certains cas. Cela peut s'expliquer par le fait que les mesures sont effectuées toutes les secondes, et donc la limite entre un état normal et un état anormal peut être très fine. Il y a également des échantillons mal classés appartenant à la classe **scan** qui est sous-représentée, ce qui peut expliquer les difficultés rencontrées par les modèles pour les classer correctement. De même, l'étiquette **scan** est plusieurs fois prédite de manière erronée, ce qui montre bien que les modèles ont du mal à savoir à quoi

correspond cette classe.

Entrée	Label réel	Nombre d'erreurs	Prédiction majoritaire
36	scan	10	normal
768	normal	10	physical fault
848	normal	10	physical fault
927	normal	10	MITM
1257	normal	10	physical fault
1549	scan	10	normal

TABLE 9 – Échantillons physiques systématiquement mal classés lors des différentes exécutions

Entrée	Label réel	Nombre d'erreurs	Prédiction majoritaire
TO COMPLETE	TO COMPLETE	TO COMPLETE	TO COMPLETE

TABLE 10 – 20 échantillons réseau systématiquement mal classés lors des différentes exécutions

Conclusion

Références

- [1] Simone GUARINO et al. *A hardware-in-the-loop water distribution testbed (WDT) dataset for cyber-physical security testing*. 2021. DOI : [10.21227/rbvf-2h90](https://doi.org/10.21227/rbvf-2h90). URL : <https://dx.doi.org/10.21227/rbvf-2h90>.
- [2] Li TAO et al. « Reduction of Intercarrier Interference Based on Window Shaping in OFDM RoF Systems ». In : *IEEE Photonics Technology Letters* 25.9 (mai 2013), p. 851-854. DOI : [10.1109/LPT.2013.2252335](https://doi.org/10.1109/LPT.2013.2252335). URL : <https://ieeexplore.ieee.org/document/9526562/>.
- [3] SCIKIT-LEARN DEVELOPERS. *scikit-learn Documentation*. Accessed : 2026-01-20. URL : <https://scikit-learn.org/stable/>.
- [4] IMBALANCED-LEARN DEVELOPERS. *imbalanced-learn Documentation*. Accessed : 2026-01-20. URL : <https://imbalanced-learn.org/stable/>.

Retour personnel : CERISARA Nathan

Contribution

Dans ce projet, j'ai principalement travaillé à développer l'architecture modulaire dont je suis très fier pour l'entraînement et le test des modèles, et le script qui va automatiquement analyser les résultats et les afficher dans un rapport markdown et les mettre à disposition de l'app streamlit. J'ai entraîné les modèles sur mon ordinateur fixe (Ryzen 5 3600, 24 Go de RAM, GTX Titan X) et j'ai surveillé les entraînements et modifié certaines configurations pour faire que certains modèles apprennent mieux et réduire l'overfitting.

Avis

J'ai pas appris grand chose avec ce projet, je savais déjà analyser / traiter des données, et je savais déjà mettre en place une pipeline d'entraînement de modèles. Si on avait eu le temps (pas eu à faire 7 projets en même temps dont 2 très gros et très importants), j'aurais peut-être pu aller plus loin dans le concept de prédiction d'attaque et de détection d'anomalie, et de mettre en place des méthodes que l'on n'utilise que dans ces domaines, mais je n'ai donc pas eu le temps de regarder cela.

Par contre, j'ai relevé un gros problème dans le papier associé au dataset, il n'y pas de code fourni accompagnant le papier avec toutes les indications pour pouvoir exactement reproduire les données du papier. On a donc par exemple un doute sur s'ils ont fait un F1 score macro ou un F1 score weighted. Personnellement, je ne comprends pas du tout comment un papier sans code public peut être accepté dans une grande conférence (comme IEEE) sans fournir un code qui permet aux autres chercheurs de comparer et de comprendre pourquoi ils n'ont pas les mêmes résultats que ceux du papier, et on ne peut donc aussi absolument pas vérifier qu'ils n'aient pas triché d'une manière ou d'une autre.

Retour personnel : DESBERG Clément

Contribution

Je me suis occupé exclusivement de la partie streamlit. Je n'ai absolument pas l'habitude de présenter des données sous la forme d'une application web, je trouvais donc cette partie particulièrement intéressante. J'ai demandé à mes camarades de me donner des fonctions dont les résultats seraient les données à afficher dans un fichier python précis (qui faisait en quelque sorte office d'API interne). À partir des deux fonctions de récupération des données, j'ai donc réalisé l'application streamlit, en prenant soit d'exclure toutes les informations fournies qui ne me semblaient pas pertinentes voir préjudiciables, et en intégrant les autres données de manière intelligibles.

Avis

Bien que je n'ai pas écrit le code des tests des modèles ou de nettoyage des données d'entrée, j'ai quand même appris un certain nombre de choses, d'abord sur streamlit en lui-même, mais aussi sur la manière dont on pouvait présenter les résultats d'un tel projet autrement que dans un rapport (je n'avais jamais ne serait-ce qu'entendu parler de streamlit avant ce projet). J'ai beaucoup aimé streamlit et la manière de l'utiliser pour présenter notre projet, c'est agréable à utiliser, très user-friendly (le fait de pouvoir afficher directement un nombre assez grand de types de données différents, et le support des icônes rondes de google pour les menus), et très pratique en terme de développement pur (le fait de ne pas avoir à relancer l'app à chaque modification, au prix d'un recalcul complet des graphiques à chaque rechargement de la page ou chaque bouton cliqué). Je n'ai pas de point négatif à soulever par rapport à ce projet.

Retour personnel : JACQUET Ysée

Contribution

Il faut bien au moins une personne pour le rapport, et c'est moi. Installation Latex (parce qu'Overleaf ne permet pas de travailler à plus que 2), rédaction, mise en forme... Pour être honnête, j'aurais aussi aimé faire de l'analyse de données (généralement la partie que je préfère) mais les données prenaient une place trop conséquente pour mon ordinateur. La découverte de Streamlit m'intéressait également, mais Clément s'en est très bien occupé (et je maîtrisais Latex, pas lui). Donc c'était une question d'optimisation de l'efficacité du groupe, je dirais.

Avis

C'est typiquement mon genre de projet préféré, dommage qu'il tombe au milieu d'une période chargée (mes camarades en parlent suffisamment dans leur paragraphe). Je pense que l'analyse, dans le sens l'explication, des données et des résultats auraient pu être plus poussés avec plus de temps. Je notifierai juste de penser aux ordinateurs qui n'ont plus beaucoup de place (10Go de données, c'est peut-être un peu too much pour du scolaire). En tout cas je pars du principe que si le projet n'est pas fait sur des heures de cours, alors on doit pouvoir le réaliser sur notre ordinateur personnel, ce qui veut dire pas nécessairement beaucoup de place ou de puissance.

Retour personnel : LEVY Lucas

Contribution

Dans ce projet, j'ai fait une première analyse sur le fichier Network pour faire une vérification des labels et des colonnes de `attack_1.csv`, à la fois pour les distributions et le nombre de vides selon les colonnes. J'ai ensuite développé un premier random forest sur les données network pour obtenir un premier jet de modèle. J'ai complété l'analyse en la déployant sur tous les fichiers Network et j'ai étendu l'analyse aux fichiers Physical afin de la rendre globale. J'ai rajouté des choses dans l'analyse qui permettent d'obtenir de nouvelles distributions comme par exemple la distribution des protocoles par labels. Enfin, j'ai exporté les résultats de l'analyse en JSON pour l'affichage streamlit et en TXT pour un visuel plus simple et pour aider le transfert vers streamlit qui n'a pas été utilisé pour cela mais ça m'a aussi permis de mieux vérifier les erreurs.

Avis

J'ai peu appris de choses nouvelles lors de ce projet en tant que compétence technique, cependant, j'ai bien aimé l'analyse des données car les données sont vraiment réelles et on doit contourner un certain nombre de problèmes humains qui se retrouvent dans un vrai dataset. Je suis content du résultat mais probablement que j'aurais pu aller plus loin ou être plus propre si le nombre de projets en parallèle de celui-ci n'était pas aussi grand.