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

Table des figures

Liste des tableaux

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 traffic réseau en période normale
`phy_norm.csv`	Acquisitions des mesures physiques en période normale
`attack_1.csv`	Première acquisition du traffic réseau avec anomalies
`dataset_att_1.csv`	Première acquisition des mesures physiques avec anomalies
`attack_2.csv`	Deuxième acquisition du traffic réseau avec anomalies
`dataset_att_2.csv`	Deuxième acquisition des mesures physiques avec anomalies
`attack_3.csv`	Troisième acquisition du traffic réseau avec anomalies
`dataset_att_3.csv`	Troisième acquisition des mesures physiques avec anomalies
`attack_4.csv`	Quatrième acquisition du traffic 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.

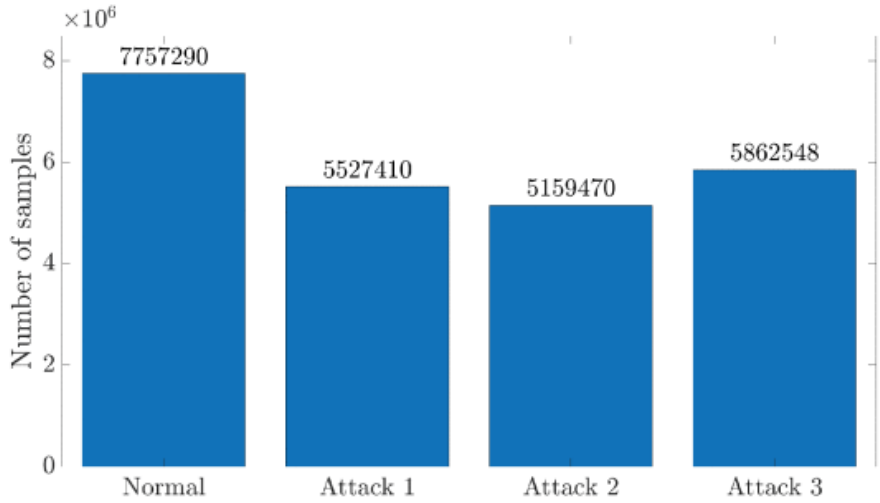

FIGURE 2 – Nombre d'entrées par fichier de données réseau d'après [2]

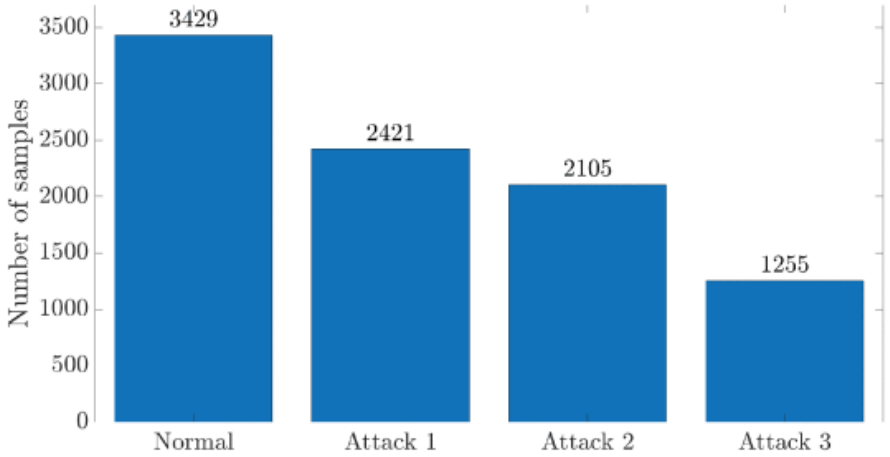

FIGURE 3 – Nombre d'entrées par fichier de données physiques d'après [2]

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.

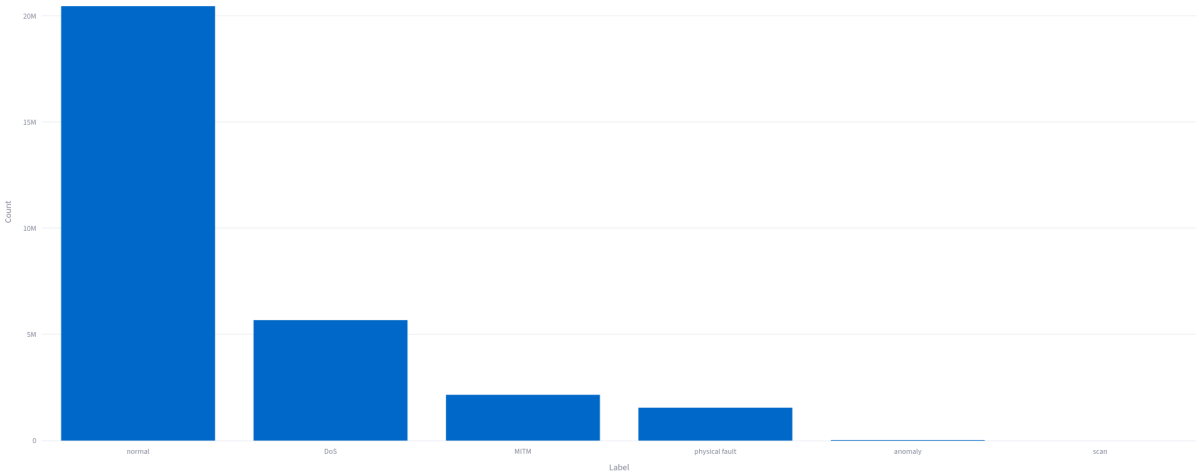

FIGURE 4 – Répartition des types d'attaques dans les données réseau

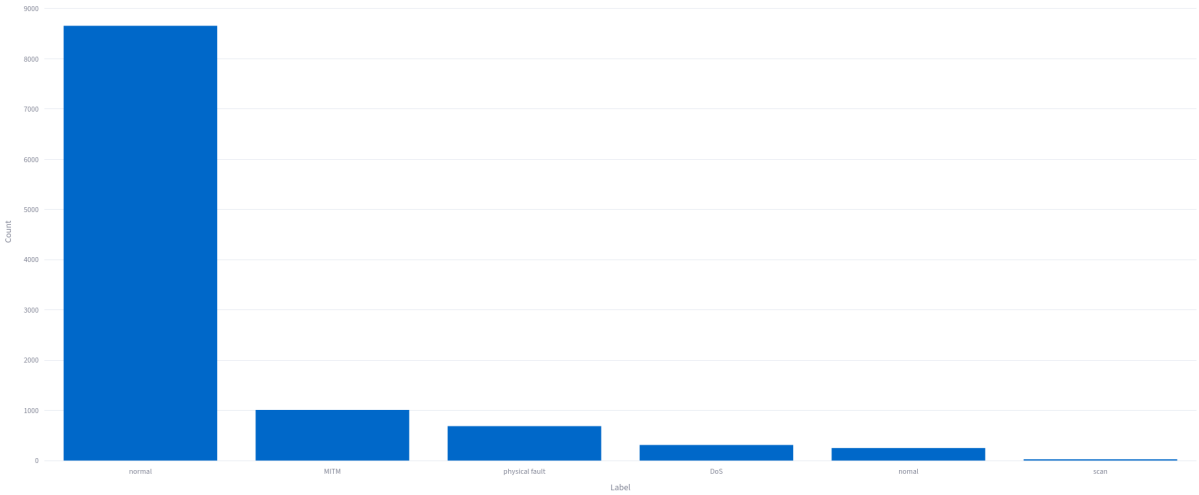

FIGURE 5 – Répartition des types d'attaques dans les données physiques

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 dificiles 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 addresse source durant les 2 dernières secondes
14	`num_pkts_dst`	Nombre de paquets de la même addresse 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 presion 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

1.2 Pré-traitement

L'ensemble des données est soumis a 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 labels sont encodés avec `LabelEncoder` de `scikit-learn` [3], ce qui permet de convertir les étiquettes textuelles en valeurs numériques.
5. 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.
6. 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].
7. 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 Hyperparamères 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 donnée 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/report.md` 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 classés sur l'ensemble des modèles, avec la prédiction majoritaire,
- une matrices 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 Données réseau

3.2 Données physiques

D'après les résultats présentés dans la Table 5, le modèle `small_knn` obtient les meilleures performances sur le jeu de données `physical_small`, avec une précision de **97,74%**, un F1-macro de **0,8007**, une précision équilibrée de **0,8235** et un MCC de **0,9402**. Le temps d'entraînement est également très faible (quasi-instantané), ce qui en fait un choix efficace pour ce type de données. À l'inverse, le modèle `small_random_forest` affiche les performances les plus faibles, ce qui peut s'expliquer par une répartition des classses TO COMPLETE.

Rang	Modèle	Précision	F1 (macro)	Précision équilibrée	MCC	Temps (s)
1	`small_knn`	0.9774	0.8007	0.8235	0.9402	0.0
2	`small_tab_transformer`	0.9683	0.8001	0.8250	0.9188	40.7
3	`small_attention_mlp`	0.9048	0.7259	0.8106	0.7946	35.2
4	`small_mlp`	0.8786	0.6889	0.7988	0.7515	22.7
5	`small_ft_transformer`	0.8401	0.6593	0.7970	0.7004	56.1
6	`small_xgboost`	0.8676	0.6412	0.7906	0.7287	0.6
7	`small_random_forest`	0.6101	0.4687	0.7242	0.4779	0.2

TABLE 5 – Comparaison des expériences sur le jeu de données physiques physical_small

En comparant les résultats obtenus avec ceux de l'article de référence [2], on constate que nos modèles TO COMPLETE. En effet dans la Table 6, on peut voir que les performances des KNN, Random Forest, SVM et Naive Bayes (NB) sont TO COMPLETE.

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

En s'attardant sur les erreurs commises par les différents modèles, on peut observer certaines corrélation entre celles-ci, comme le montre la Figure 4. Par exemple — et sasns surprise — le modèle MLP et le modèle MLP avec attention présentent une forte corrélation dans leurs erreurs.

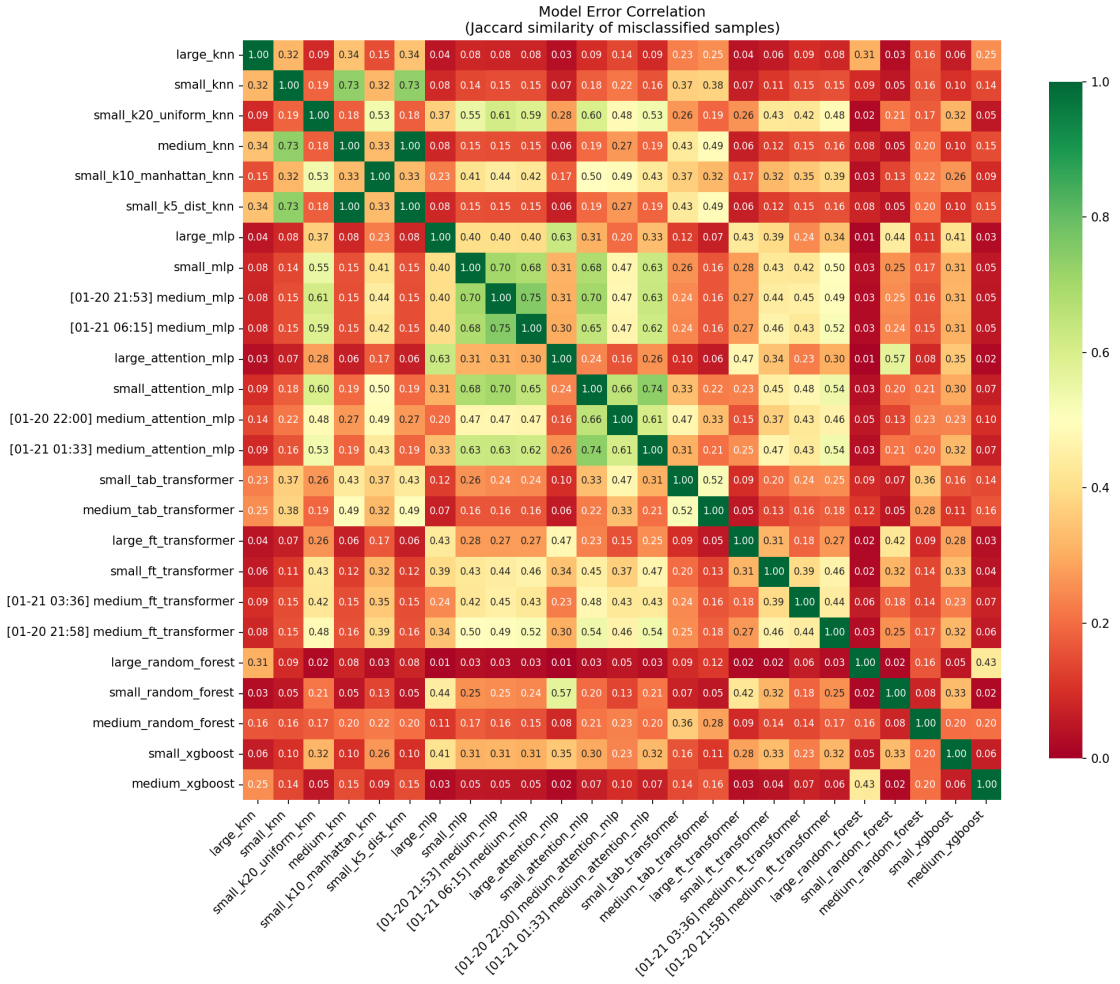

FIGURE 6 – Matrice de corrélation des erreurs entre les modèles pour les données physiques

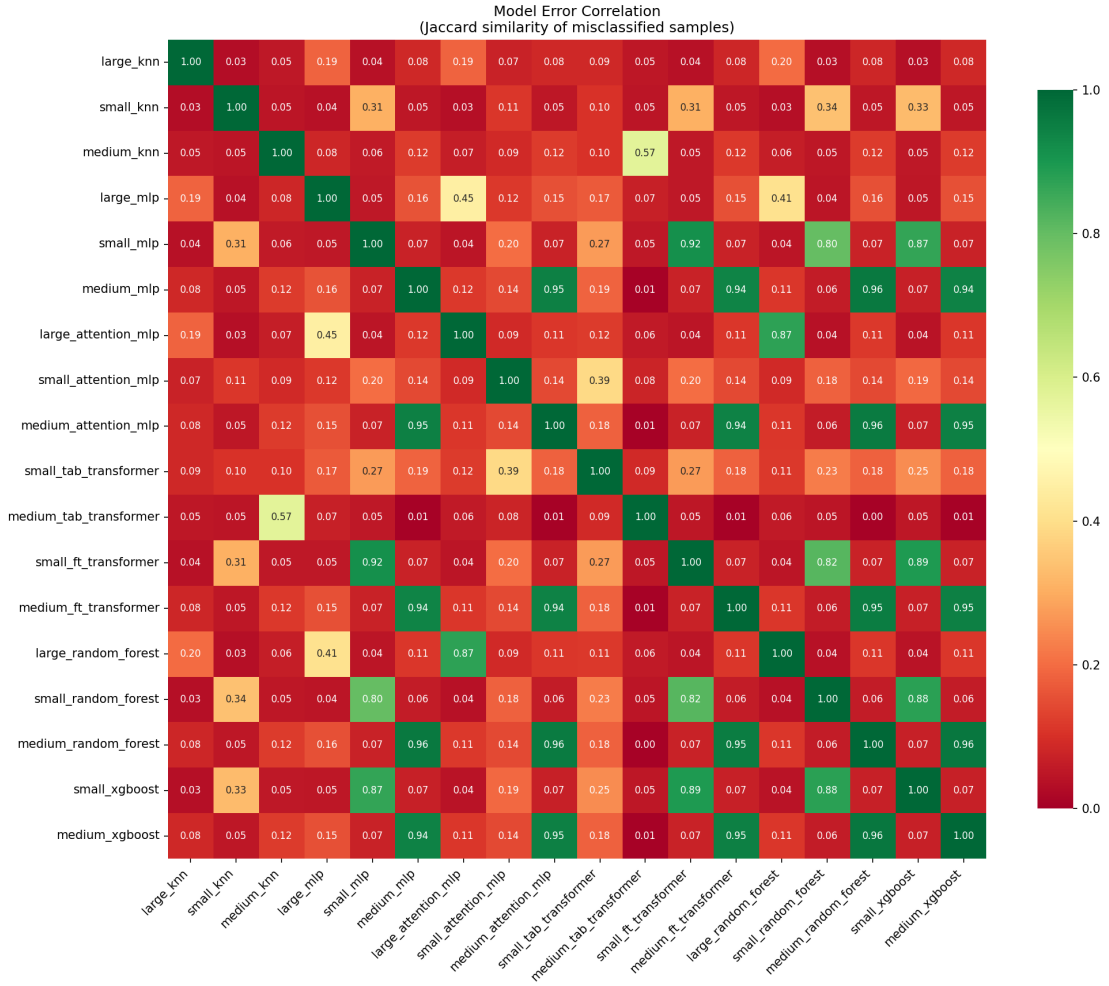

FIGURE 7 – Matrice de corrélation des erreurs entre les modèles pour les données réseau

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 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 7 – 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 8 – 20 échantillons réseau systématiquement mal classés lors des différentes exécutions

Conclusion

Retour personnel : CERISARA Nathan

Contribution

Dans ce pojet, 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.

Retour personnel : DESBERG Clément

Contribution

Avis

Retour personnel : JACQUET Ysée

Contribution

Avis

Retour personnel : LEVY Lucas

Contribution

Avis

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. 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. 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/.