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

1 Données **1**
1.1 Visualisation des données 1
1.1.1 Présentation générale des données 1
1.1.2 Analyse des données 4
1.2 Prétraitement 4

2 Application des algorithmes **4**
2.1 Algorithmes utilisés 4
2.2 Organisation des entraînements 5
2.3 Hyperparamères et résultats 5

3 Évaluation et comparaison avec les données de référence **6**
3.1 Données physiques 6

Références **9**

Références **9**

Table des figures

1 Nombre d'entrées par fichier de données physiques d'après [2] 2
2 Nombre d'entrées par fichier de données réseau d'après [2] 3
3 Matrice de corrélation des erreurs entre les modèles 7

Abréviations employées

- **CART** : Classification And Regression Trees
- **CNN** : Convolutional Neural Network
- **CSV** : Comma Separated Values
- **FT Transformer** : Feature Tokenizer Transformer
- **KNN** : K-Nearest Neighbors
- **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. Parmi les modèles demandés, K-Nearest Neighbors (KNN), Random Forest, XGBoost et Multi-Layer Perceptron (MLP) ont été implémentés. De plus, des modèles basés sur des architectures de transformers adaptées aux données tabulaires on été testés en remplacement de l'algorithme de Classification And Regression Trees (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.

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 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 à une anomalie (étiquettes `DoS`, `MITM`, `scan` pour les données réseau et `physical fault` pour les données physiques).

Données physiques

En s'appuyant sur les informations fournies par [2], on peut dénombrer les nombre de lignes pour chaque jeu de données physique comme présenté dans la Figure 1.

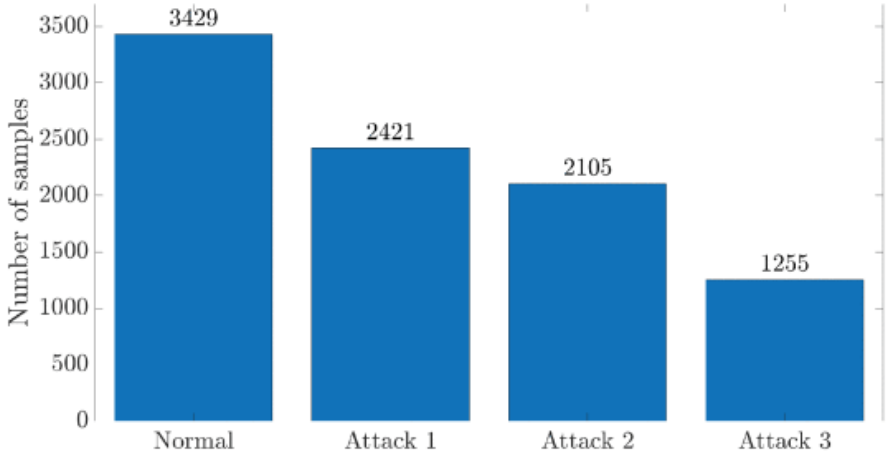

FIGURE 1 – Nombre d'entrées par fichier de données physiques d'après [2]

On constate que le fichier `phy_norm.csv` contient un total de **3 429** entrées, tandis que les fichiers d'attaques contiennent entre 1 2000 et 2 421 entrées chacun, et sont donc 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.*

Les mesures physiques sont reportées toutes les secondes et concernent des paramètres tels que l'état des pompes, les valeurs de différents capteurs, etc. En tout, il y a 41 caractéristiques, présentées dans la Table 2 (une traduction du tableau original présent dans [2]).

Données réseau

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 2 – Caractéristiques des données physiques

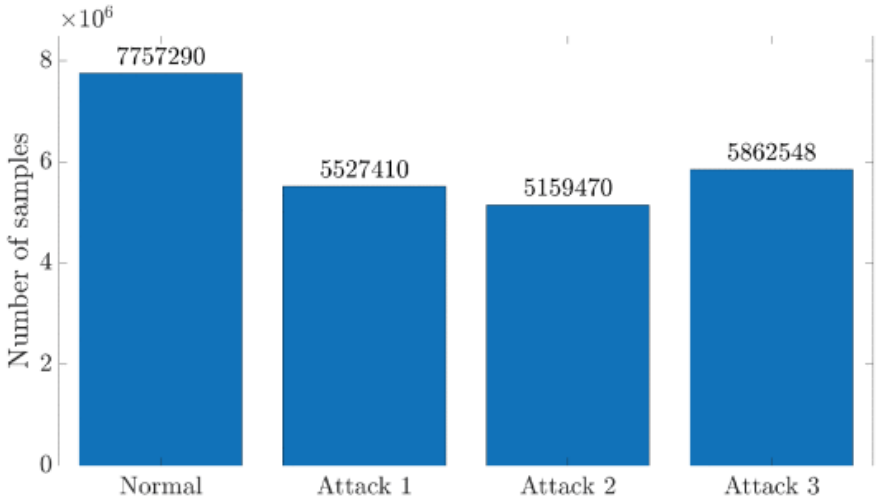

FIGURE 2 – Nombre d'entrées par fichier de données réseau d'après [2]

Au nombre de 14, les colonnes des données réseau sont 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

1.1.2 Analyse des données

1.2 Prétraitement

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` [3] de `scikit-learn` [4], 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` [5] 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` [6] de la bibliothèque `imbalanced-learn` [7].
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

Parmi les modèles demandés, les suivants ont été implémentés :

- KNN
- Random Forest
- XGBoost
- MLP
- Modèles de transformers

Le modèle CART n'a pas été implémenté, mais a été remplacé par des modèles basés sur des architectures de transformers adaptées aux données tabulaires :

- Tab Transformer : TO COMPLETE
- FT Transformer : TO COMPLETE

- MLP avec attention : 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 (physiques et réseau). Chaque modèle est défini dans un script séparé contenu dans un dossier `models/`, avec des configurations spécifiques pour chaque type de modèle. Les données sont pré-traitées de manière cohérente pour tous les modèles, et un système de cache est mis en place pour éviter de répéter le pré-traitement lors de nouvelles exécutions. Les résultats de chaque entraînement sont enregistrés de manière structurée, permettant une analyse et une comparaison faciles des performances des modèles.

2.3 Hyperparamères et résultats

- Organisation des modèles bien structurée
- Scripts définissant la structure de chaque modèle dans le dossier `models/`.
- Différentes configurations pour chaque modèle (hyperparamètres, datasets, taille des modèles, etc.) dans `configs/model_type/config_name.yaml`.
- Scripts d'entraînement et de test dans `scripts/train.py` et `scripts/test.py`.
- Les données sont pré-traitées en amont **de la même façon pour tous les types de modèles**.
- Un système de cache est utilisé pour éviter de traiter à nouveau les données lors de nouvelles exécutions et sont enregistrées dans le dossier `cached_datasets`.
- Ensuite, les modèles sont entrainés individuellement avec le script `scripts/train.py` sur un fichier de config, ou alors il est aussi possible d'entraîner tous les modèles ou un groupe de modèles d'un coup avec le script `scripts/train_all.py`.
- Les résultats de chaque entraînement de chaque config de modèle sont enregistrés dans le dossier `experiments`.
- Sont enregistrés pour chaque entraînement :
  - les poids du meilleurs modèle
  - la configuration utilisée (au cas ou les configurations changent)
  - Les métriques sur le jeu de données d'entraînement, validation et test du meilleur modèle.
  - Les erreurs sur le jeu de données d'entraînement, validation et test du meilleur modèle.

- ▶ Les courbes de loss et d'accuracy sur le jeu de donnée d'entraînement et de validation.

- Un rapport récapitulatif de tous les entraînements est ensuite généré avec le script `src/analyze_results.py` séparé pour le physical et le network datasets.

Pour les hyperparamètres, pour l'instant, ont juste été testés 2 tailles de modèles (small et medium) pour chaque dataset, et utilisés des bons learning rate et autres hyperparamètres par défaut, et de bons résultats ont été obtenus.

3 Évaluation et comparaison avec les données de référence

3.1 Données physiques

D'après les résultats présentés dans la Table 4, 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 4 – 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 5, 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 5 – 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 1. Par exemple — et sasns surprise — le modèle MLP et le modèle MLP avec attention présentent une forte corrélation dans leurs erreurs.

FIGURE 3 – Matrice de corrélation des erreurs entre les modèles

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**
5	normal	7	MITM
35	normal	7	MITM
36	scan	7	normal
210	normal	7	MITM
466	normal	7	DoS
598	normal	7	scan
768	normal	7	physical fault
848	normal	7	physical fault
864	normal	7	scan
892	normal	7	DoS
927	normal	7	MITM
1104	normal	7	MITM
1203	normal	7	scan
1257	normal	7	physical fault
1549	scan	7	normal
1559	normal	7	physical fault
1628	normal	7	MITM

TABLE 6 – Échantillons 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. 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. *sklearn.preprocessing.LabelEncoder*. Accessed : 2026-01-19. 2025. URL : https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html.

[4] F. Pedregosa et al. « Scikit-learn : Machine Learning in Python ». In : *Journal of Machine Learning Research* 12 (2011), p. 2825-2830. URL : http://www.jmlr.org/papers/v12/pedregosa11a.html.

[5] scikit-learn developers. *sklearn.model_selection.train_test_split — scikit-learn documentation*. Accessed : 2026-01-19. 2025. URL : https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[6] imbalanced-learn developers. *imblearn.over_sampling.RandomOverSampler — imbalanced-learn documentation*. Accessed : 2026-01-19. 2025. URL : https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html.

[7] Guillaume Lemaître, Fernando Nogueira et Christos K. Aridas. « Imbalanced-Learn : A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning ». In : *Journal of Machine Learning Research* 18.17 (2017), p. 1-5. URL : http://jmlr.org/papers/v18/16-365.html.