Explorations des IA pour IDS sur bus CAN

Evain Arthur

1er janvier 2026

Résumé

A travers ce stage, j'ai exploré trois modèles d'IA qui avaient pour but d'identifier un trafic anormal sur un bus CAN :

— CANet [3], qui a une architecture composé d'un réseau de neurone récurrent par type de paquet et d'une structure d'autoencodeur
— CAN-BERT [1], qui se base sur la structure de transformeur BERT pour prédire une séquence d'ID de paquets consécutifs
— ADVision, qui se base sur deux réseaux concurrents, un discriminateur (CNN) et un générateur

J'ai pu entraîner ces modèles sur les données du protocole SAEJ1939 collectées lors d'un projet précédent. Je les ai ensuite intégrés au code de l'IDS développé actuellement par Secure-IC. J'ai enfin commencé à les intégrer au projet d'installation d'IDS sur le BagXone d'Alstef.

1 Introduction

1.1 Contexte

Plus de détails sont donnés dans les papiers de recherche [3] [1]

Les bus CAN sont utilisés dans le domaine automobile pour permettre la communication entre les différents terminaux électroniques (ECU) sans multiplier les câbles. Chaque ECU voulant communiquer envoie un message en broadcast (c'est-à-dire à tous les autres ECUs) où se trouve son ID unique et un payload contenant des informations (signaux) uniques en nombre et en contenu.
Ce système de communication, bien que pratique, manque de sécurité, d'un système d'authentification par exemple. C'est pourquoi les IDS (systèmes de détection d'intrusion) se sont développés pour pallier à ce problème. Ces systèmes écoutent le trafic en temps réel et lancent une alerte en cas d'anomalie.
De nombreux IDS se basent sur des classifieurs comme Adaboost qui apprennent à classifier un comportement normal ou anormal pour chaque signal indépendamment. Cette méthode est efficace pour détecter des attaques comme *flooding* (injection à haute fréquence de paquets) ou *fuzzy* (injection aléatoire de paquets) mais échoue sur des attaques plus sophistiquées comme *replay* (remplacement des paquets par une séquence de paquets émis précédemment sur le bus). Pour cause, chaque modèle étudie son signal indépendamment des autres, là où comprendre la relation entre les ECU et les signaux est essentielle.
J'ai donc exploré des modèles qui sont capables d'analyser l'intégralité du trafic.

1.2 Attaques

Les attaques étudiées ont lieu pendant une durée limitée et visent à perturber le système. Elles sont plus ou moins faisables selon le niveau de connaissances ou d'accès que l'attaquant a vis-à-vis du trafic (par exemple, un attaquant qui peut contrôler intégralement certains ECUs

et qui sait déchiffrer chaque paquet du réseau pourra faire des attaques plus sophistiquées) : j'ai synthétisé avec un tableau la faisabilité de chaque attaque (ce tableau est juste indicatif et n'est basé sur aucun travail autre que le mien). Ces attaques n'ont pas pu être réalisées en vrai : j'ai seulement modifié les données de trafic normal pour les ajouter, ce qui n'est pas très réaliste car le système aurait réagi en conséquence.
Pour réaliser une attaque, l'attaquant doit au moins être capable d'émettre un paquet ou d'empêcher un ECU d'émettre (par exemple en l'éteignant). Les attaques décrites peuvent parfois être réalisées de deux manières : soit en injectant un paquet juste après chaque émission de paquet de l'ECU ciblé (parfois mentionné comme targetted ID attack), soit en interceptant chaque paquet de l'ECU ciblé avant qu'il ne soit émis et en émettant le paquet d'attaque à la place (masquerade attack).
Voici la liste des attaques étudiées :

- Plateau : valeur du signal fixée à une constante (souvent au maximum possible)
- Continuous change : valeur du signal modifiée progressivement jusqu'à une valeur (il s'agit d'une variante plus "douce" de l'attaque plateau)
- Replay : remplace le trafic par celui ayant eu lieu précédemment. On enregistre le trafic à un moment donnée, et on rejoue ce trafic à la place du vrai trafic pendant l'attaque. Cette attaque est peut être la plus importante à considérer car elle est efficace, ne nécessite aucune connaissance préalable sur le trafic et est difficile à repérer. Elle a déjà été mise en oeuvre par Secure-IC lors de précédents projets.
- Flooding : injection de paquets à haute fréquence (généralement plus élevée que la normale)
- Suppress : suppression des paquets émis par un ECU (typiquement réalisé par l'extinction de l'ECU)
- Fuzzy : injection de paquets aléatoires (aussi bien pour l'ID que pour les champs de données)

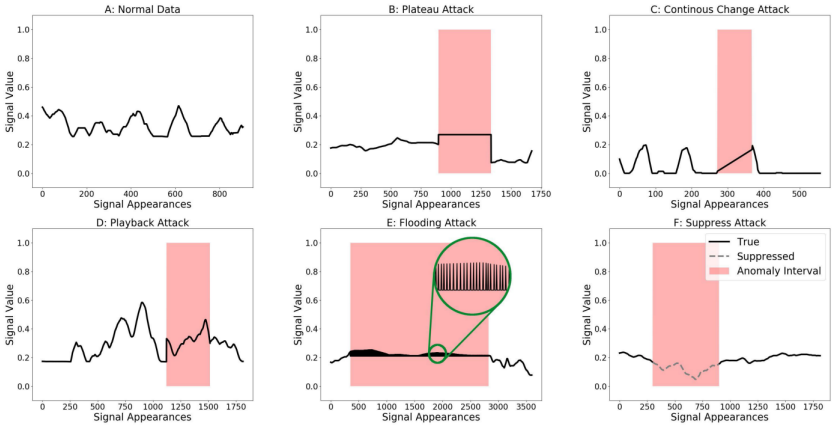

FIGURE 1 – Attaques étudiées pour CANet (source : [3])

Type d'attaque	Accès	Connaissances
Plateau	Emission	ID
Continuous change	Emission	ID + déchiffrement
Replay	Emission	Traffic
Flooding	Emission	Aucune
Suppress	Contrôle d'un ECU	Aucune
Fuzzy	Emission	Aucune

TABLE 1 – Récapitulatif des attaques et de leur faisabilité

2 Modèles

Le principe est le même pour les trois modèles : ceux-ci sont entraînés sur des données de traffic normal (c'est-à-dire non attaqué). En situation réelle, s'ils se trompent trop, c'est que le traffic présente une anomalie.
Les trois modèles se sont révélés complémentaires :

- CANet [3] analyse les données du paquet et peut donc détecter les attaques modifiant le payload (mais reposant sur une décryption du payload),
- CAN-BERT [1] n'analyse que les IDs des paquets, ce qui permet de détecter des injections anormales de paquets, même si on ne dispose d'aucune information sur le payload,
- ADVision n'analyse que le contenu des paquets. Il fait des prédictions peu risquées : il réussit donc a détecter tout type d'attaque régulièrement avec peu de faux positifs.

Voici une rapide description de leur fonctionnement (se référer aux papiers pour approfondir).

2.1 CANet

CANet [3] se base sur une architecture d'autoencodeur, qui est constituée :

- d'un encodeur qui a pour but de réduire la dimension des données d'entrée jusqu'au *bottleneck*
- d'un décodeur qui essaye de reconstruire la donnée d'entrée le plus fidèlement possible en partant du *bottleneck*

On peut penser à l'autoencodeur comme deux entonnoirs successifs. CANet prend en entrée un paquet du bus CAN : c'est une donnée très hétérogène, car le nombre de variables et leur signification varie selon l'ID de l'ECU qui a envoyé le paquet. C'est pourquoi CANet possède un réseau de neurones récurrent LSTM pour chaque ID : chaque paquet entrant est donné à ce LSTM, dont la sortie est concaténée à celle de tous les autres LSTM pour former un vecteur latent. C'est ce vecteur qui sera ensuite donné en entrée de l'autoencodeur.
On comparera la sortie de l'autoencodeur correspondante au paquet entrant en calculant la distance quadratique entre chaque donnée du paquet. Si cette distance est trop grande, cela voudra dire que le paquet est anormal.
Quelques précisions techniques :

- chaque LSTM étend la sortie d'un facteur `h_scale` par rapport à l'entrée, ce qui représente intuitivement la capacité computationnelle du modèle
- l'encodeur est constitué d'une couche cachée de dimension deux fois inférieure à la couche d'entrée
- le *bottleneck* ne réduit que de 1 la dimension de l'entrée
- le décodeur n'a pas de couche cachée

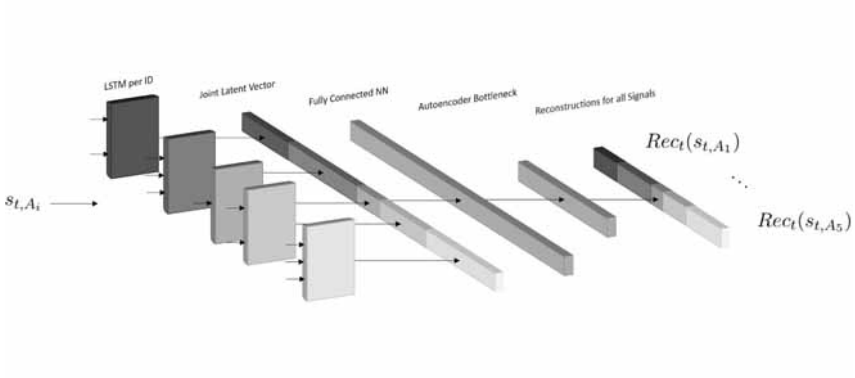

FIGURE 2 – Architecture de CANet

2.2 CAN-BERT

CAN-BERT [1] se base sur le modèle de transformeur BERT. Son vocabulaire est constitué de tous les IDs d'ECU sur le bus CAN, qui sont convertis en tokens (numéros de 1 jusqu'au nombre total d'IDs différents). Le modèle prend en entrée une séquence d'IDs consécutifs dont un certain nombre est masqué (par un caractère spécial de son vocabulaire *MASK*), et son but est de deviner quels sont les IDs derrière chaque *MASK*. Il s'appuie pour cela sur le contexte bi-directionnel (à la différence de GPT).
Son architecture est celle d'un transformeur de type BERT : il encode la valeur et la position de chaque ID, puis le contexte par un mécanisme d'attention à une ou plusieurs têtes succédé d'un mécanisme de *position-wise feedforward*, qui applique un perceptron à chaque token (à savoir que l'application de ces deux mécanismes se fait 4 fois dans le modèle). Enfin, le vecteur est reprojeté dans son espace initial.
Quelques précisions techniques :

- 45% des tokens sont remplacés par *MASK* sur chaque séquence
- les séquences sont de taille 64
- les sorties du modèle sont passées dans un softmax, dont l'image est à interpréter comme le niveau de confiance en chaque ID
- le modèle a le droit à 5 essais pour chaque *MASK*

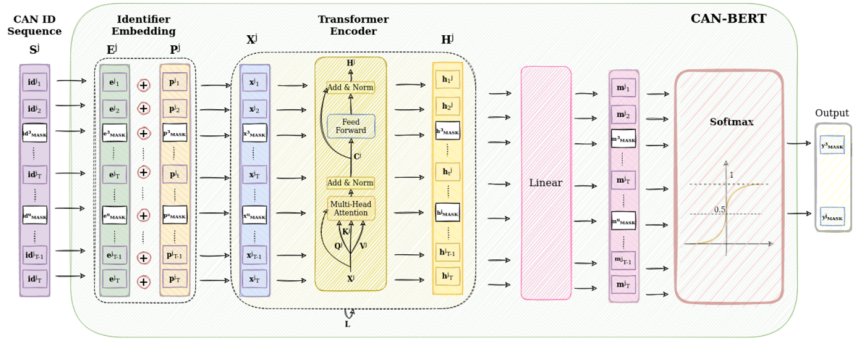

FIGURE 3 – Architecture de CAN-BERT

2.3 ADVision

ADVision se base sur la conversion d'une séquence de paquets en image. Chaque ligne de l'image correspond aux signaux du paquet, et plusieurs paquets sont ainsi mis les uns au-dessus des autres pour créer une "image" rectangulaire. Les paquets n'ayant pas tous le même nombre

de signaux, on remplit les trous avec des 1 (arbitrairement).
ADVision est composé de deux modèles concurrents :

— un Discriminateur, qui est un CNN qui détermine si une image correspond à un trafic anormal ou non
— un Générateur, qui est un perceptron qui essaye de créer une image à partir de bruit la plus réaliste possible pour tromper le Discriminateur

Quelques précisions techniques :

— Le Générateur doit être bien plus entraîné que le Discriminateur, au moins d'un facteur 10.

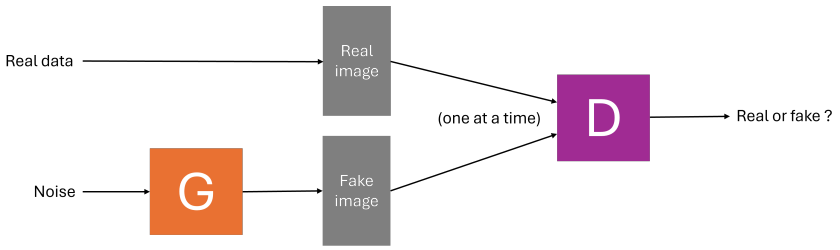

FIGURE 4 – Fonctionnement de ADVision

3 Entraînement

Cette section s'intéresse à l'entraînement et à l'obtention des hyperparamètres des modèles, c'est-à-dire des caractéristiques décrivant l'architecture des modèles qui ne sont pas modifiées par l'entraînement. Cette analyse permet de se faire une intuition sur les performances des modèles.

3.1 Données

Le papier de recherche de CANet [3] ne fournissait publiquement que des données synthétiques. Celles-ci contenaient 10 ECUs qui s'échangeaient un total de 20 signaux, sur 30 millions de lignes de données. Les attaques étaient générées elles aussi synthétiquement. Au final, ces données permettaient au modèle d'atteindre un haut niveau de précision.
Le papier de recherche de CAN-BERT [1] utilisait les données du "In Vehicle Network Intrusion Detection Challenge" [4]. Celles-ci contiennent 100 COBID différents.
Les données réelles que j'ai utilisées (SAEJ1939) contiennent 28 COBID différents. On ne s'intéresse qu'à 3 IDs qui envoient un total de 12 signaux, avec de plus grandes disparités dans la fréquence d'apparition de chaque ECU.

3.2 Méthodes

L'entraînement a été géré par Pytorch. Voici les détails (approximatifs) des paramètres d'entraînement (qui sont proches de ceux des papiers) :

`num_epochs`	200
`learning_rate`	0.01
`batch_size`	5000
`chunk_size`	250
`weight_decay`	10^{-5}

TABLE 2 – Paramètres d'entraînement pour CANet

num_epochs	200
learning_rate	0.01
dropout	0.1

Table 3 – Paramètres d'entraînement pour CAN-BERT

num_epochs	200
learning_rate	0.01
generator_factor	13

Table 4 – Paramètres d'entraînement pour ADVision

Quelques détails d'entraînement :

— j'ai mis un scheduler de facteur 2 et de patience 10 sur le learning rate (il était divisé par 2 si aucune meilleure perte n'avait été enregistré sur les 10 derniers batches)

— j'ai ajouté de la standardisation aux modèles : weight decay pour CANet et dropout pour CAN-BERT. Ces paramètres permettent une balance entre erreur d'entraînement et erreur de généralisation

L'idée générale était d'entraîner de cette façon sur un même nombre d'époques les modèles avec des hyperparamètres différents, et d'ensuite comparer les métriques calculées pendant l'entraînement. La librairie Python optuna permet d'automatiser ce processus.

CANet performe très inégalement selon les signaux. Sa performance sur chaque batch pouvait donc beaucoup varier (du simple au double !) à un même stade de l'entraînement. C'est pourquoi je collectais la perte sur de multiples batchs et que je comparais les moyennes et variances sur ceux-ci.

3.3 Régularisation

La régularisation désigne les techniques qui permettent d'améliorer la capacité des modèles à généraliser leurs résultats des données d'entraînement aux données de test. Ces méthodes ne sont pas non plus magiques : elles "handicapent" les modèles et réduisent leurs performances sur données d'entraînement. Un compromis (sous forme d'hyperparamètres à ajuster) doit donc être trouvé. Voici les techniques que j'ai utilisé dans mes modèles :

— Dropout : désactivation de certains neurones

— Weight decay : pénalité sur les trop grandes valeurs de poids

— Batch normalization (pas retenue mais intéressante à mentionner) : normalisation des batchs pour qu'ils aient une moyenne à 0 et une variance à 1

3.4 Hyperparamètres

Voici les hyperparamètres que j'ai retenu (colonnes Expérience). Il faut retenir que le choix de ceux-ci doit être adapté à chaque traffic CAN. Les paramètres d'entraînement sont aussi des hyperparamètres dont la modification doit être considérée au cas où les hyperparamètres sont modifiés.

Hyperparamètre	Papier	Expérience
h_scale	10	25
encoder_layers	1	2
decoder_layers	1	2

Table 5 – Comparatif des hyperparamètres pour CANet

Hyperparamètre	Papier	Expérience
`d_model`	256	256
`num_layers`	4	4
`num_heads`	1	16
`mask_ratio`	0.45	0.6

Table 6 – Comparatif des hyperparamètres pour CAN-BERT

Hyperparamètre	Expérience
`latent_dim`	50
`img_height`	100

Table 7 – Hyperparamètres pour ADVision

4 Résultats

Voici un détail des performances de chaque modèle sur les diverses attaques que j'ai présenté dans la section Introduction. Les graphiques ont en ordonnée le numéro d'apparition du signal et en abscisse la prédiction du modèle (normalisée entre 0 et 1). Les pointillées rouges indiquent la présence ou non d'une attaque : à 0, le trafic est normal, ailleurs il est attaqué. Enfin, les courbes bleues ou vertes indiquent la prédiction du modèle : plus la prédiction est proche de 0, plus le trafic peut être considéré comme normal (i.e. moins le modèle s'est trompé).
Il faut garder en tête que ces attaques n'ont pas de "réalité physique" : elles ont été générées manuellement à partir de relevés de données réelles. Dans la réalité, les paquets non attaqués réagiraient à la présence d'une attaque, et les résultats seraient donc différents.

4.1 CANet

La prédiction de CANet est la distance quadratique entre sa reconstruction et le traffic réel. Le modèle s'est vite relevé performant pour détecter tout type d'attaque suffisamment grossière sur le payload. Les attaques plateau et continuous change sont repérées quand elles ont lieu. Cependant, les attaques ne sont repérées que lors des transitions (lorsqu'elles apparaissent ou disparaissent) : je pense que cela n'aurait pas lieu sur une vraie attaque, les autres signaux "s'affolant" en réaction aux grandes valeurs.

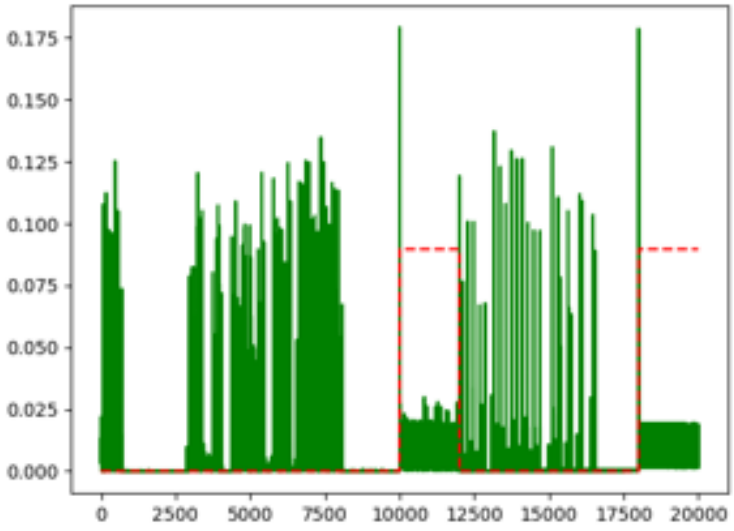

FIGURE 5 – Prédictions sur plateau attack

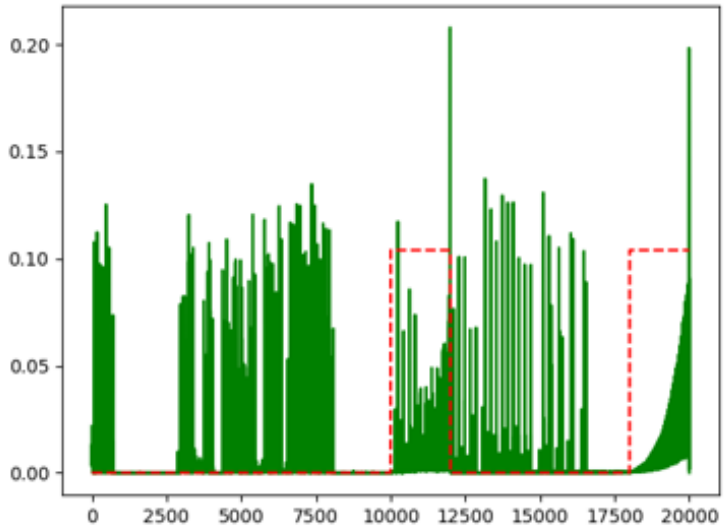

FIGURE 6 – Prédictions sur continuous change attack

L'attaque replay est plus difficilement repérées. Le problème réside dans le fait que le modèle n'est pas assez précis sur les données normale, et donc la différence est difficile à faire entre une vraie alerte et une erreur intrinsèque aux performances du modèle.

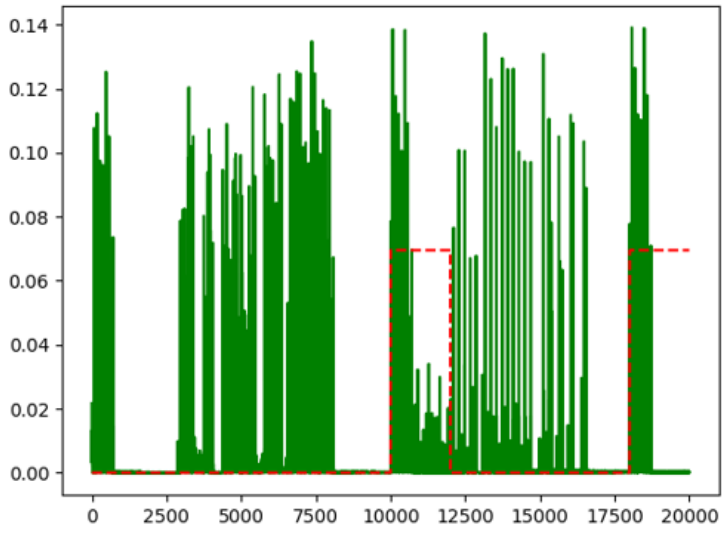

FIGURE 7 – Prédictions sur replay attack

Globalement, CANet est efficace et sera retenu pour les attaques visant le payload. Comme les autres modèles, son intérêt réside aussi dans sa capacité à repérer des attaques jamais observées auparavant. Notre attention va maintenant se porter sur la détection des replay attacks.

4.2 CAN-BERT

La prédiction de CAN-BERT est le nombre de tokens MASK pour lesquels le modèle n'a pas mis le bon token parmi ses 5 candidats les plus probables.
Ce modèle est le plus satisfaisant en termes de prédiction des replay attacks. Attention tout de même aux erreurs de prédiction sur trafic normal : comme pour CANet, un compromis doit être trouvé entre précision sur valeur d'entraînement et capacité de généralisation (voir section Régularisation).

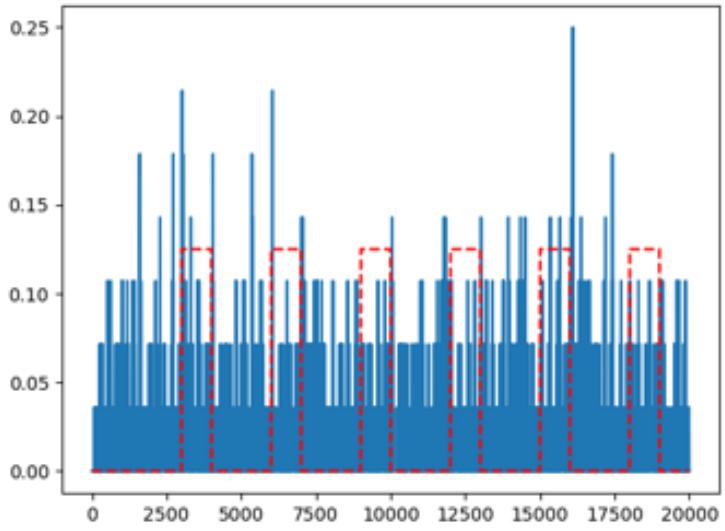

FIGURE 8 – Prédictions sur replay attack

4.3 ADVision

La prédiction d'ADVision est la sortie du réseau Discriminateur.
Ce modèle, cette fois-ci, est très "sûr de lui" et prédit des valeurs très petites. Par contre, quand il prédit des valeurs plus hautes, on peut être assez sûr qu'une attaque a eu lieu. Il ne produit donc aucun faux positif (au prix de beaucoup de faux négatifs).

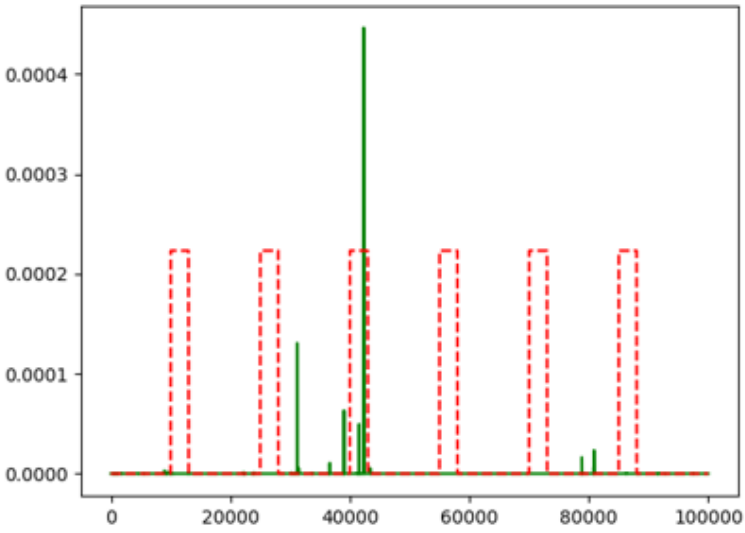

FIGURE 9 – Prédictions sur replay attack

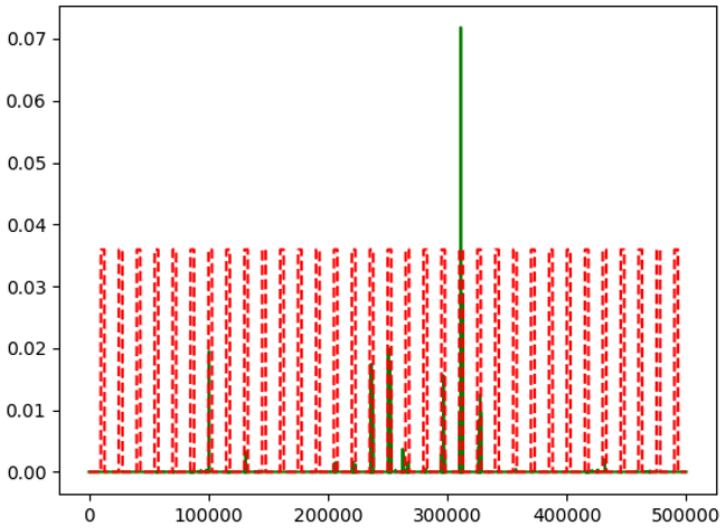

FIGURE 10 – Prédictions sur plateau attack

Ce modèle entraîne aussi un générateur à comprendre le trafic. Il peut potentiellement créer des attaques complexes pour entraîner CANet.

5 Discussion

Une autre approche différente peut être considérée : une approche statistique basée sur un HMM [2]. L'idée est cette fois de considérer le trafic comme un flux de données atomiques, en abandonnant la structure de paquets.
Il faudrait utiliser des métriques plus reconnues pour publier dans un papier de recherche (F1-score, AUC...). On pourrait aussi analyser l'importance des features et les relations entre signaux/IDs.
Voici des pistes à explorer pour les différents modèles (si besoin) :

- Pour CANet : tenter d'étendre encore plus la surface de l'encodeur (et du décodeur, car on préfère garder de la symétrie) ; adapter h_scale selon le signal (si celui-ci est plus complexe / que le modèle galère à bien le fit).
- Pour CAN-BERT : changer le critère comme quoi le modèle se trompe si le bon token n'est pas parmi les 5 avec la plus grande probabilité
- Pour ADVision : changer les paramètres du CNN, exploiter le générateur

De plus, je n'exploitais pas très bien les capacités des cartes graphiques : adapter/optimiser le code pour les utiliser serait sûrement grandement bénéfique à l'entraînement. Notamment, cela permettrait d'entraîner CAN-BERT sur l'intégralité du dataset. Aussi, réussir à faire traiter des batches à CANet (sans regarder chaque élément individuellement) serait une grande amélioration.

6 Conclusion

Références

[1] Natasha Alkhatib, Maria Mushtaq, Hadi Ghauch, and Jean-Luc Danger. Can-bert do it ? controller area network intrusion detection system based on bert language model. In

2022 IEEE/ACS 19th International Conference on Computer Systems and Applications (AICCSA), pages 1–8, 2022.

[2] Chen Dong, Hao Wu, and Qingyuan Li. Multiple observation hmm-based can bus intrusion detection system for in-vehicle network. *IEEE Access*, 11 :35639–35648, 2023.

[3] Markus Hanselmann, Thilo Strauss, Katharina Dormann, and Holger Ulmer. Canet : An unsupervised intrusion detection system for high dimensional can bus data. *IEEE Access*, 8 :58194–58205, 2020.

[4] Hyunjae Kang, Byung Il Kwak, Young Hun Lee, Haneol Lee, Hwejae Lee, and Huy Kang Kim. Car hacking : Attack defense challenge 2020 dataset, 2021.