RAPPORT DE PROJET DE MASTER

R.A.I.D.

RAM Artificial Inspection Dump

1er juin 2026

HENRY Amand — ROUSÉE Tom — SICOT Théo

Supervisé par :
GIGUET Emmanuel — BENOIT Tristan

Résumé

Le projet R.A.I.D. (RAM Artificial Inspection Dump) explore l'application de l'apprentissage profond à l'analyse forensique de la mémoire vive (RAM). Face aux limites des méthodes traditionnelles, souvent dépendantes de signatures statiques et vulnérables à l'obfuscation ou à la fragmentation des données, ce travail propose une approche innovante basée sur l'architecture Transformer.

L'objectif central est de développer un système capable de classifier de manière autonome les octets d'un dump mémoire pour déterminer leur nature (chiffrée ou non chiffrée, type de contenu comme texte, PDF ou image). Pour ce faire, nous avons conçu un pipeline complet incluant la génération de datasets synthétiques réalistes et l'implémentation du modèle `BytesTransformerClassifier`. Ce modèle hybride, combinant une couche de convolution locale à un encodeur *Transformer*, permet d'extraire des dépendances contextuelles complexes au sein de flux binaires bruts.

Nos résultats démontrent l'efficacité de cette méthode, atteignant une précision globale de 94,79% grâce à l'implémentation d'un mécanisme de vote pondéré visant à stabiliser les prédictions. Bien que le modèle présente une confusion résiduelle entre les données chiffrées et compressées deux types de données à haute entropie une phase de fine-tuning ciblé a permis d'équilibrer les performances et de renforcer significativement la capacité de discernement du classifieur. Ce projet ouvre des perspectives prometteuses pour l'automatisation de l'analyse forensique, suggérant que l'IA peut transformer radicalement la manière dont les preuves volatiles sont extraites et interprétées.

Table des matières

1 Introduction **3**

1.1 Contexte . . . 3

1.2 Problématique . . . 3

1.3 Objectifs . . . 5

2 Méthodologie **6**

2.1 Générateur de datasets synthétiques . . . 6

2.1.1 Collecte de données . . . 6

2.1.2 Transformation des données . . . 6

2.1.3 Création du segment RAM synthétique . . . 7

2.1.4 Construction des différents dumps . . . 8

2.2 Le découpage en chunks . . . 9

2.3 Modèle de classification . . . 9

2.3.1 Introduction au principe du *Transformer* . . . 9

2.4 `BytesTransformerClassifier` . . . 12

2.5 Entraînement et validation . . . 14

2.6 Inférence . . . 15

2.6.1 Processus d'inférence . . . 15

2.6.2 Principe de *recouvrement* (l'overlapping) . . . 16

2.7 Optimisation . . . 16

2.8 Outils d'analyse . . . 18

3 Résultats **21**

3.1 Le vote pondéré . . . 21

3.2 Comparaison des configurations du modèle . . . 21

3.3 Performance Globales . . . 22

3.3.1 Les performances du modèle avant fine tuning . . . 22

3.3.2 Performances du modèle après fine-tuning . . . 24

4 Discussion **27**

5 Conclusion **32**

Bibliographie **35**

1 Introduction

1.1 Contexte

L'analyse forensique de la mémoire vive (RAM) constitue un outil de l'informatique légale. Contrairement au stockage persistant, la RAM contient des artéfacts volatils critiques : clés de chiffrement, fragments de documents, ou résidus de textes souvent invisibilisés dans de grandes quantités de données déstructurées.

Jusqu'aux années 2000, aucun outil n'était conçu pour l'analyse, les experts se reposaient sur des méthodes ad hoc, et utilisaient des outils génériques comme `strings` ou `grep` pour récupérer les chaînes de caractères. Par la suite, pendant la première décennie du siècle, des outils comme *Volatility* ou *Rekall* ont émergé, offrant des capacités d'extraction plus avancées basées sur des signatures statiques.

Cependant, l'extraction d'informations reste complexe. Bien que ces outils aient progressé, les méthodes traditionnelles reposent souvent sur ces signatures statiques, qui sont à la fois rigides et vulnérables à l'obfuscation. De plus, la nature fragmentée des dumps mémoire rend difficile l'identification de structures de données critiques, sans compter la présence de données chiffrées ou compressées qui présentent une entropie élevée et peu de motifs exploitables.

Dans ce contexte, l'approche par apprentissage automatique permet de trouver des règles sémantiques dans les séquences binaires. Il y a quelques années, le travail de Thomas Gougeon *« Analyse forensique de la mémoire des cartes à puce » (2017)* a démontré la capacité d'un RNN (réseau de neurones récurrents) à différencier les données chiffrées des données textuelles dans le contexte spécifique des cartes à puce. Cependant, le passage à l'échelle pour traiter des dumps mémoire complets, avec leur diversité de formats et de structures, reste un défi ouvert.

1.2 Problématique

L'analyse forensique de la mémoire vive s'est historiquement structurée autour de la recherche de signatures statiques et de structures rigides (comme les en-têtes de fichiers ou

les structures de données des systèmes d'exploitation via des outils comme *Volatility*). Face à la fragmentation et à l'obfuscation, ces approches montrent leurs limites.

Pour dépasser ce paradigme rigide, des travaux antérieurs ont exploré l'apport de l'apprentissage automatique. C'est le cas notamment de la thèse de Thomas Gougeon, *« Analyse forensique de la mémoire des cartes à puce » (2017)*. Il a démontré la viabilité de modèles pour différencier des données chiffrées de données textuelles. Cependant, son approche repose sur une analyse qui s'apparente à une approche par signatures statistiques de haut niveau, car elle s'appuie sur l'extraction de features calculées sur des blocs de données (comme le calcul explicite de l'entropie ou de fréquences de caractères). De plus, ces travaux se limitent à des environnements contraints comme les cartes à puce et ne se confrontent pas au passage à l'échelle qu'imposent des dumps mémoire complets et hétérogènes, ceux-ci mélangeant encodages, compressions multiples et données bruitées.

Le projet R.A.I.D. propose de passer d'une analyse par signatures ou caractéristiques précalculées à une analyse profonde basée sur des motifs que l'humain pourrait peiner à pointer du doigt. En exploitant une architecture *Transformer*, le modèle n'a plus besoin qu'on lui définisse ce qu'est l'entropie : il doit apprendre de manière autonome la topologie et les dépendances contextuelles complexes au sein d'une séquence binaire brute, simulant une véritable compréhension "syntaxique" du binaire.

Néanmoins, en l'absence d'une comparaison expérimentale directe et exhaustive avec l'ensemble des méthodes traditionnelles ou les travaux précédents de Thomas Gougeon (faute de benchmarks standardisés disponibles), une question fondamentale se pose. La problématique de ce travail s'articule donc ainsi :

Dans quelle mesure une architecture d'apprentissage profond de type Transformer, exploitant uniquement les relations contextuelles locales et globales d'une séquence binaire brute, permet-elle de segmenter efficacement un dump de RAM hétérogène, et ce, même en l'absence de descripteurs explicites ou de signatures traditionnelles ?

Autrement dit, il s'agit de déterminer si les méthodes d'apprentissage profond sont capables de capturer intrinsèquement la sémantique des données pour surmonter un point de difficulté de l'analyse forensique, notamment la distinction entre haute entropie "utile" (compression) et haute entropie "intentionnelle" (chiffrement).

1.3 OBJECTIFS

Le projet R.A.I.D. (RAM Artificial Inspection Dump) s'inscrit dans cette dynamique en proposant une approche basée sur l'apprentissage profond. L'objectif central est de développer un système capable de classifier les octets d'un dump de RAM pour déterminer leur nature : chiffrée ou non, et ce de manière autonome.

L'implémentation repose sur l'utilisation de l'architecture *Transformer* et en particulier l'encodeur ainsi que son mécanisme de *self-attention* associé.

L'hypothèse de travail est qu'un modèle de langage, entraîné à classer des séquences d'octets, sera capable de différencier les données chiffrées, des données compressées, par nature à forte entropie, tout en identifiant facilement les structures de données plus conventionnelles et prévisibles, telles que du texte brut.

Cette différenciation doit permettre à terme de segmenter précisément la mémoire et d'identifier les différents champs de données sans supervision humaine.

2 Méthodologie

2.1 Générateur de datasets synthétiques

2.1.1 Collecte de données

Pour constituer notre corpus initial, nous avons agrégé des données issues de trois sources principales :

— des textes bruts provenant du *projet Gutenberg*

— des documents PDF via *ArXiv*

— des images via *Lorem Picsum*

Le *projet Gutenberg* est une bibliothèque numérique de plus de 60 000 livres du domaine public, offrant une grande diversité de textes littéraires. *ArXiv*, quant à lui, fournit un accès à des milliers de documents scientifiques au format PDF. Et enfin, *Lorem Picsum* propose une vaste collection d'images. Les trois services offrent des API permettant de récupérer facilement des données de manière automatisée, ce qui a permis de constituer un corpus initial.

2.1.2 Transformation des données

Afin de transformer cet échantillon en un jeu de données réaliste et exploitable, ces fichiers bruts sont soumis à un processus de transformations reproduisant la diversité des données en mémoire.

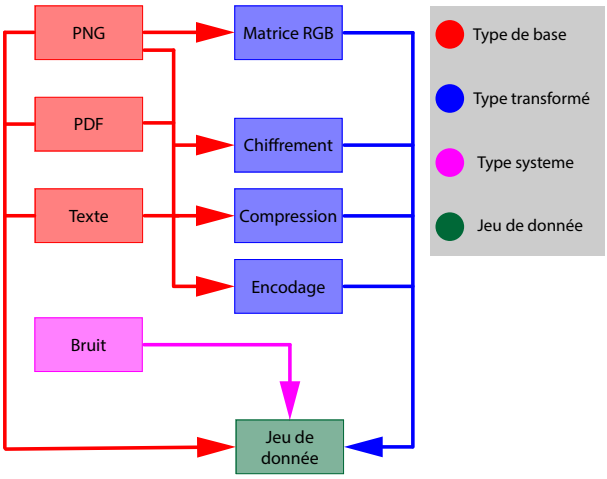

FIGURE 2.1 – Schéma du processus de transformation des données

Nos trois types de données principaux (Texte, PDF, Images) sont soumis à des transformations détaillées dans la Figure 2.1, telles que l'encodage via le `Base64`, la compression à taux variable via `zLib` ou `Gzip`, le chiffrement via l'algorithme `AES-256-CBC`, ou encore la conversion en matrice de pixels RGB pour les images. Également, nous ajoutons du bruit aléatoire. Toutes ces transformations sont appliquées avec une graine afin de garantir la reproductibilité du processus tant que les données sources restent inchangées.

Avec toutes ces transformations, nous obtenons une grande variété de sous-classes de données, que nous étiquetons en fonction de leur nature, chiffrées ou non chiffrées. Cette diversité est essentielle pour entraîner un modèle performant, qui devra faire face à des données fragmentées, obfusquées ou structurées de manière imprévisible. Nos deux classes sont donc réparties comme montré dans le Tableau 2.1.

Non chiffrée (0)	Chiffrée (1)
Texte brut PDF Image brute (PNG) Image convertie en matrice de pixels RGB Données compressées via `zLib` et `gLib` Données encodées via `Base64` Données bruitées	Données chiffrées

TABLE 2.1 – Répartition des sous-classes de données dans le jeu de données synthétique

2.1.3 Création du segment RAM synthétique

Tout le processus de création du dump est reproductible grâce à l'utilisation d'une graine, garantissant que les mêmes données sources génèrent le même dump synthétique à chaque exécution du processus. La répartition des différentes classes est équilibrée, avec 50% de données chiffrées et 50% de données non chiffrées, afin d'éviter tout biais lors de l'entraînement du modèle. Chaque type des données non chiffrées est également représenté de manière équilibrée pour garantir une diversité suffisante au sein de cette classe, illustré dans la Figure 2.2.

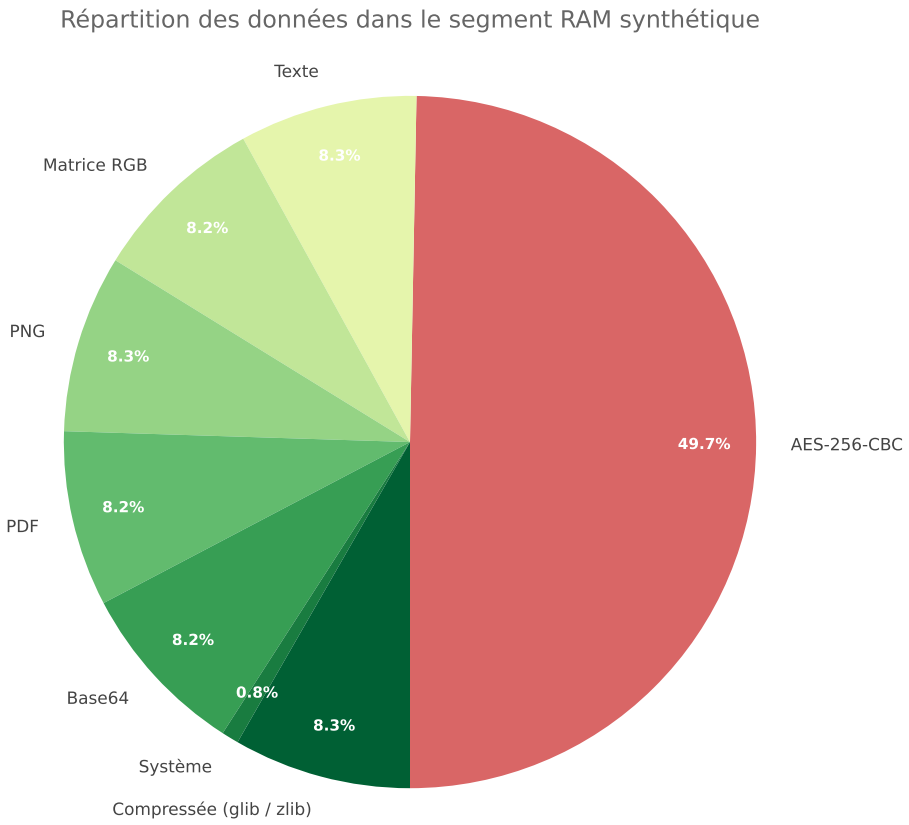

FIGURE 2.2 – Répartition des données dans le dump synthétique

Une fois transformées, les données sont injectées dans un espace mémoire contigu, le dump synthétique. Chaque fichier est découpé en segments de taille variable, entre 1024 et la taille totale du fichier, afin de simuler la fragmentation naturelle des données en RAM. Les segments sont ensuite mélangés de manière aléatoire pour éviter toute corrélation spatiale entre les différentes classes, ce qui rend la tâche de classification plus réaliste et complexe pour le modèle.

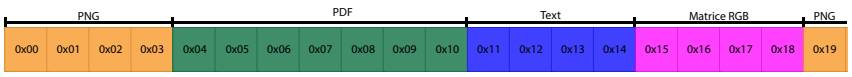

FIGURE 2.3 – Illustration d'un dump RAM synthétique

2.1.4 Construction des différents dumps

Avec cette méthode de génération de données, nous avons construit deux dumps synthétiques de 1,2 Go et 500 Mo, dédiés respectivement à l'entraînement et au test du modèle. Ces dumps sont équilibrés, avec la répartition de la Figure 2.2. Pour le dump d'entraînement,

20% des données sont réservées à la validation, afin de suivre l'évolution de la performance du modèle et d'éviter le surapprentissage. Le dump de test, quant à lui, est utilisé pour évaluer la capacité du modèle à généraliser.

2.2 LE DÉCOUPAGE EN CHUNKS

Pour permettre au modèle de traiter efficacement les données, le dump est découpé en segments contigus de taille fixe, appelés *chunks*. Chaque *chunk* contient 1024 octets, ce qui correspond à la taille maximale que le modèle peut traiter en une seule passe. Ce découpage permet de structurer les données pour permettre au modèle d'apprendre à identifier des motifs locaux au sein de chaque segment. Cependant, ce découpage en *chunks* introduit une discontinuité artificielle dans les données, car les frontières entre les *chunks* peuvent couper des structures de données ou des motifs importants. Pour compenser cette limitation, nous avons implémenté une stratégie de recouvrement lors de l'inférence, qui sera détaillée dans la Sous-section 2.6.2.

2.3 MODÈLE DE CLASSIFICATION

2.3.1 Introduction au principe du *Transformer*

L'architecture *Transformer* représente un changement de paradigme majeur dans le domaine de l'apprentissage profond (Deep Learning). Introduit par *Google* en 2017 à travers l'article *« Attention is All you Need » (2017)*, ce modèle s'est imposé comme le standard pour le traitement de données séquentielles, notamment dans le traitement automatique des langues (NLP).

Le *Transformer* repose sur un principe de parallélisation massive des flux de données, ce qui permet de traiter des séquences entières simultanément, sans dépendre de l'ordre d'entrée. Le cœur de cette architecture réside dans le mécanisme d'attention, qui permet au modèle de pondérer l'importance relative de chaque élément d'une séquence par rapport à tous les autres, facilitant ainsi la capture de dépendances à longue distance.

Architecture Encodeur

Un encodeur traite et modélise l'information d'entrée en convertissant des données brutes en représentations vectorielles de haute dimension. Lorsqu'il est utilisé indépen-

damment d'un décodeur, ce module est particulièrement efficace pour les tâches de classification ou de compréhension sémantique, où la génération de texte n'est pas requise.

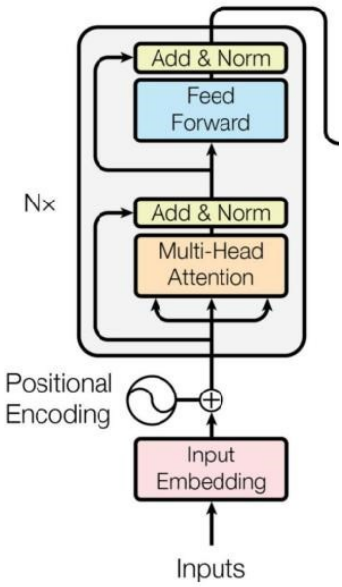

FIGURE 2.4 – Illustration d'un encodeur de l'article *« Attention is All you Need » (2017)*

Input Embedding

L'input embedding transforme les données brutes en représentations vectorielles de dimension d_{model}, rendant l'information traitable mathématiquement par le modèle.

Une fois projetés dans l'espace vectoriel, ces vecteurs sont multipliés par un facteur d'échelle $\sqrt{d_{model}}$ afin de stabiliser les gradients lors de l'entraînement.

Contrairement aux réseaux récurrents (RNN), l'architecture *Transformer* traite les éléments d'une séquence simultanément. Pour compenser cette absence de notion d'ordre intrinsèque, un encodage de position est ajouté à chaque vecteur d'entrée. Cette information permet au modèle de distinguer la position relative de chaque élément dans la séquence.

Cet encodage repose sur des fonctions sinusoïdales de fréquences différentes, définies par les formules suivantes :

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.1}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.2}$$

Mécanisme d'attention

Le mécanisme de *self-attention* permet au modèle de pondérer l'importance relative de chaque élément d'une séquence par rapport à tous les autres au sein de la même entrée. En projetant chaque token dans trois espaces distincts (Q, K, V), le modèle calcule une distribution de scores par produit scalaire. Cette opération permet d'extraire les dépendances contextuelles, qu'elles soient locales ou à longue distance, en focalisant le traitement sur les parties les plus informatives de la séquence.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.3}$$

$$Q = \text{matrice de requêtes}$$

$$K = \text{matrice de clés}$$

$$V = \text{matrice de valeurs}$$

$$d_k = \text{dimension des clés}$$

L'attention multi-têtes est l'application de plusieurs mécanismes de *self-attention* en parallèle, dans le but de spécifier et d'extraire plusieurs métriques (types, syntaxe ...) afin de ne pas écarter de sens exploitable. Leurs résultats sont ensuite fusionnés pour extraire un unique vecteur portant toutes les informations sur le sens.

Sortie du modèle

À la sortie du modèle, nous trouvons des logits (une valeur entre $-\infty$ et $+\infty$), nous y appliquons une fonction sigmoïde pour transformer les valeurs entre 0 et 1, entre les deux classes.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

2.4 BytesTransformerClassifier

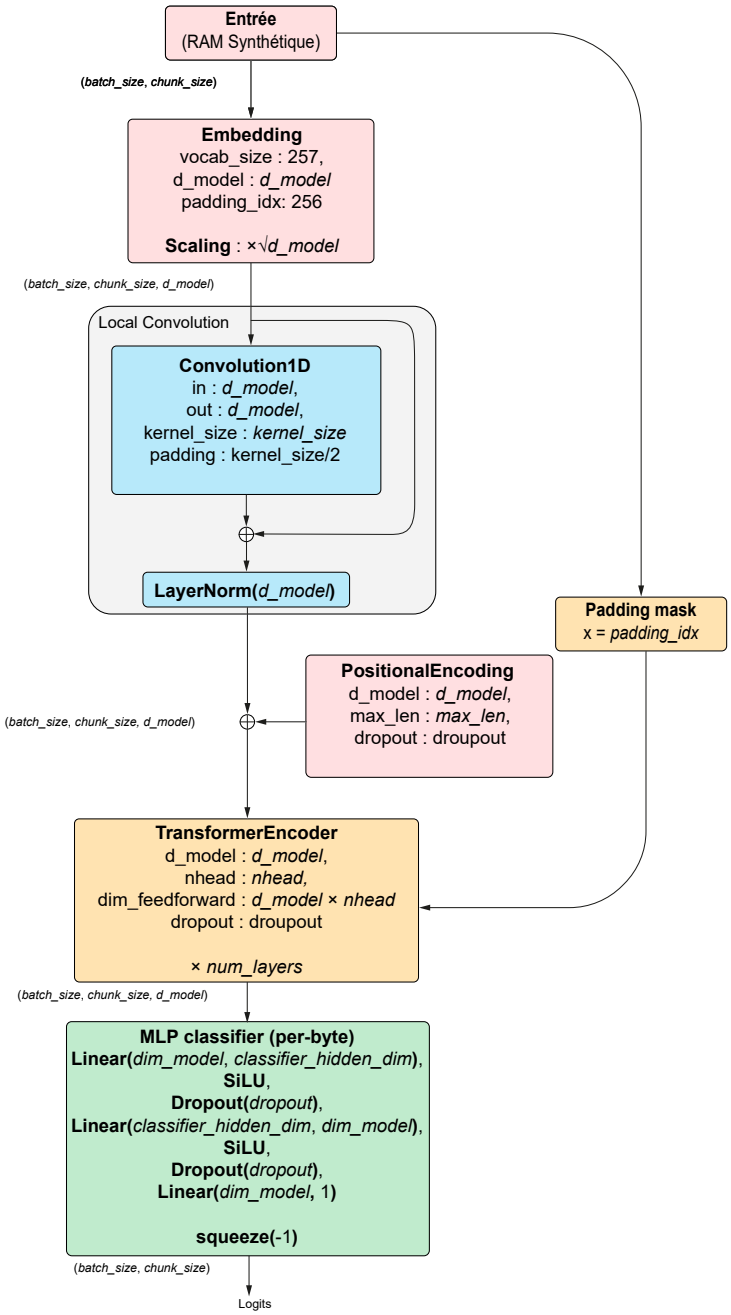

FIGURE 2.5 – Architecture du réseau : BytesTransformerClassifier

Paramètre	Valeur
chunk_size	1024
d_model	256
kernel_size	3
num_heads	4
num_layers	4
dropout	0.2
classifier_hidden_dim	256

TABLE 2.2 – Paramètres du classifieur RAID

Nous avons opté pour une architecture hybride combinant une couche de convolution unidimensionnelle (*local convolution* dans la Figure 2.5) en amont du bloc d'encodeur. Cette configuration permet d'extraire des motifs locaux au sein des séquences d'octets, tout en bénéficiant de la capacité du *Transformer* à modéliser les dépendances à longue distance.

2.5 ENTRAÎNEMENT ET VALIDATION

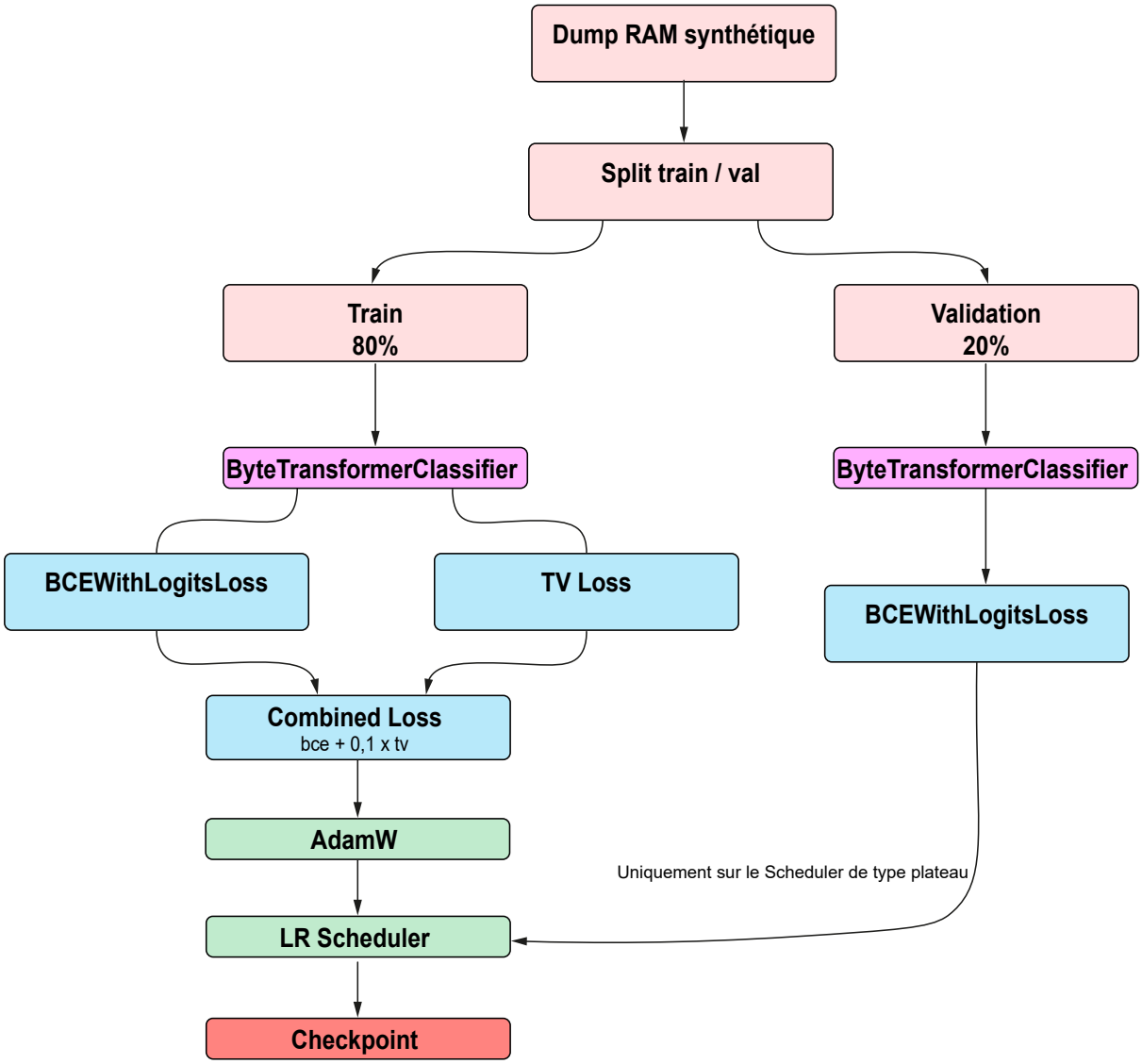

FIGURE 2.6 – Processus d'entraînement du modèle

Le dump est divisé en deux parties avec un `random_split`. 80% des données sont utilisées pour l'entraînement, tandis que les 20% restants sont réservés à la validation. Le split est reproductible grâce à l'utilisation d'une graine fixe.

Ensuite, nous calculons la loss principale avec `BCEWithLogitsLoss`, qui combine une fonction de perte binaire avec une activation sigmoïde intégrée. Nous avons également ajouté une TV loss (Total Variation Loss) pour encourager la cohérence spatiale des prédictions, en

pénalisant les changements de classe abrupts entre octets adjacents. La loss totale est une combinaison pondérée de ces deux composantes, avec un coefficient λ, ici de 0.1, ajusté pour équilibrer leur influence.

Nous utilisons l'optimiseur AdamW, qui est une variante de l'Adam classique, intégrant une régularisation par poids (weight decay) pour améliorer la généralisation du modèle, avec un taux d'apprentissage initial est fixé à $1e^{-4}$. Nous utilisons aussi un scheduler de type `CosineAnnealingLR` pour ajuster dynamiquement le taux d'apprentissage au cours de l'entraînement, permettant ainsi une convergence plus efficace. Un scheduler plateau est aussi présent mais n'a pas prouvé son efficacité. Le warmup linéaire peut être chaîné avant le scheduler principal (sauf avec Plateau) via SequentialLR. A l'issue de chaque époque, nous sauvegardons le modèle afin d'éventuellement revenir à une version antérieure si nécessaire.

Pour choisir le meilleur modèle, nous utilisons la métrique d'*accuracy* et de la *loss*. Le modèle avec l'*accuracy* la plus élevée et la *loss* la plus faible est sélectionné pour l'inférence finale.

Un des problèmes que nous avons rencontrés est la confusion entre les données chiffrées et les données compressées, nous avons procédé à une phase de fine-tuning spécifiquement ciblé pour ce problème. En réentraînant notre meilleur modèle sur trois époques strictement équilibrées (répartition 50/50 entre le chiffré et le compressé), nous avons fortement amélioré ses capacités de discernement. Les résultats sont présentés dans la Sous-section 3.3.2.

2.6 INFÉRENCE

2.6.1 Processus d'inférence

Avec le meilleur modèle sélectionné, nous procédons à l'inférence sur le dump de test. Lors de l'inférence, le dump de RAM est traité par segments de 1024 octets, conformément à la taille d'entrée du modèle. Chaque segment est passé à travers le classifieur, qui génère une prédiction binaire pour chaque octet, indiquant s'il est chiffré ou non.

2.6.2 Principe de *recouvrement* (l'overlapping)

Le modèle traite le dump mémoire par fenêtres glissantes de `N` octets. Au lieu de découper la mémoire en blocs contigus, nous utilisons un pas de glissement (stride) inférieur à la taille de la fenêtre. Par conséquent, chaque octet de la RAM est analysé plusieurs fois par le *Transformer*, au sein de contextes différents.

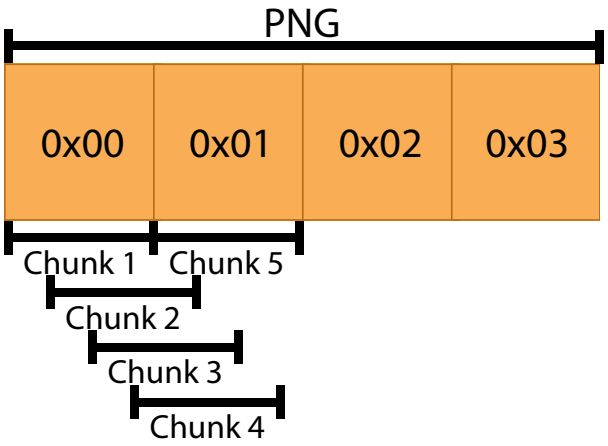

FIGURE 2.7 – Illustration du principe de l'overlapping, ici avec un stride de $\frac{1}{4}$ de chunk

Après avoir transformé les logits en probabilités via une fonction sigmoïde, les prédictions locales sont agrégées. Cette étape repose sur une pondération de chaque résultat par le score de confiance associé.

$$\text{Confiance}_x = 2 \times |p_x - 0.5| \tag{2.5}$$

$$\text{P_Global}_x = \frac{\sum_{i=1}^{n} \text{Confiance}_i \times p_i}{\sum_{i=1}^{n} \text{Confiance}_i} \tag{2.6}$$

$$\text{avec } n = \frac{\text{chunk_size}}{\text{stride}}$$

2.7 OPTIMISATION

Afin d'aller plus loin sur le projet, nous avons eu besoin d'augmenter significativement nos volumes de données synthétiques pour l'entrainement ainsi que le nombre d'époques. Nous avons donc dû trouver des optimisations afin d'exploiter le plus possible notre matériel.

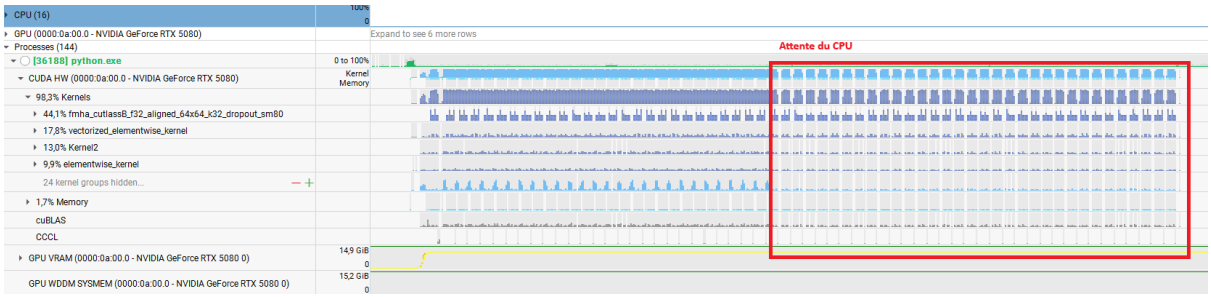

FIGURE 2.8 – Goulot d'étranglement entre le CPU et le GPU avant optimisation, visualisé avec *NVIDIA Nsight Systems*

Premièrement nous avons identifié un goulot d'étranglement entre le CPU et le GPU, comme nous pouvons le voir sur la Figure 2.8. Afin de résoudre ce problème nous avons configuré des options de parallélisation avancées sur les `Dataloaders` notamment via les `persistent_workers`, un `prefetch_factor` élevé et une refonte du code lié au `__getitem__`. Comme nous pouvons le voir sur la Figure 2.9, cela a fonctionné comme espéré.

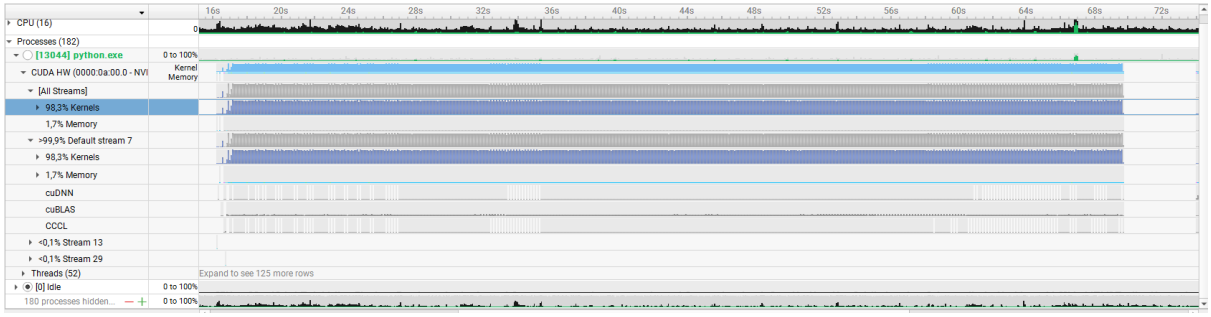

FIGURE 2.9 – Parallélisation efficace du pipeline de données après optimisation, visualisé avec *NVIDIA Nsight Systems*

Dans un second temps, nous avons maximisé la vitesse d'entraînement. Nous avons adopté la précision mixte, *bfloat16*, permettant de réduire la consommation de VRAM par deux et d'accélérer le temps d'entraînement par deux. De plus, nous avons activé `torch.compile` avec le *backend Inductor* en mode *max-autotune*. Contrairement à une approche standard, ce mode réalise un benchmark exhaustif sur notre architecture, une *RTX 5080* : le compilateur génère et exécute physiquement des variantes de noyaux Triton pour chaque segment du graphe de calcul. Cette recherche identifie la stratégie de fusion de noyaux offrant la latence la plus faible. Bien que cette phase d'*autotuning* ajoute environ 400 secondes au lancement, elle génère un gain de performance de 27,45% par époque, elle est donc intéressante sur l'entraînement mais également sur des inférences qui seraient longues.

Enfin, la Total Variation Loss a été réimplémentée sous forme d'une convolution 1D. Cette approche exploite pleinement la parallélisation massive des cœurs CUDA, rendant ce calcul quasi instantané par rapport à une approche itérative classique sur CPU.

2.8 OUTILS D'ANALYSE

Pour analyser les résultats nous avons développé un visualiseur de données, qui permet de superposer les classifications du modèle à la vérité. Il s'agit d'une interface graphique web développée en *React* avec un `<canvas>`.

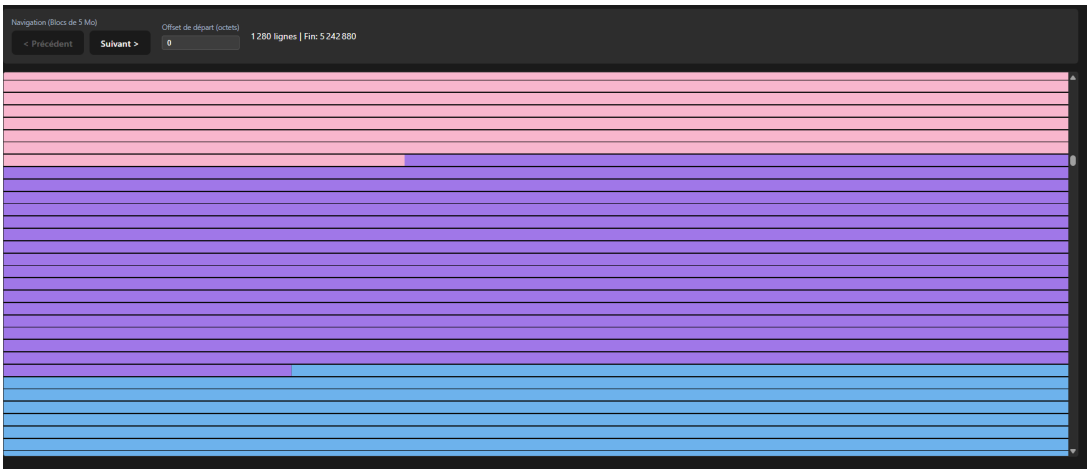

FIGURE 2.10 – Interface de visualisation des données

Le visualiseur cartographie visuellement la mémoire en attribuant une couleur spécifique à chaque type de données (PDF, image, texte, etc.), comme dans la Figure 2.10. En superposant les classifications du *Transformer* à la vérité terrain, l'outil met en surbrillance rouge les erreurs, offrant ainsi une capacité d'analyse granulaire à l'octet près.

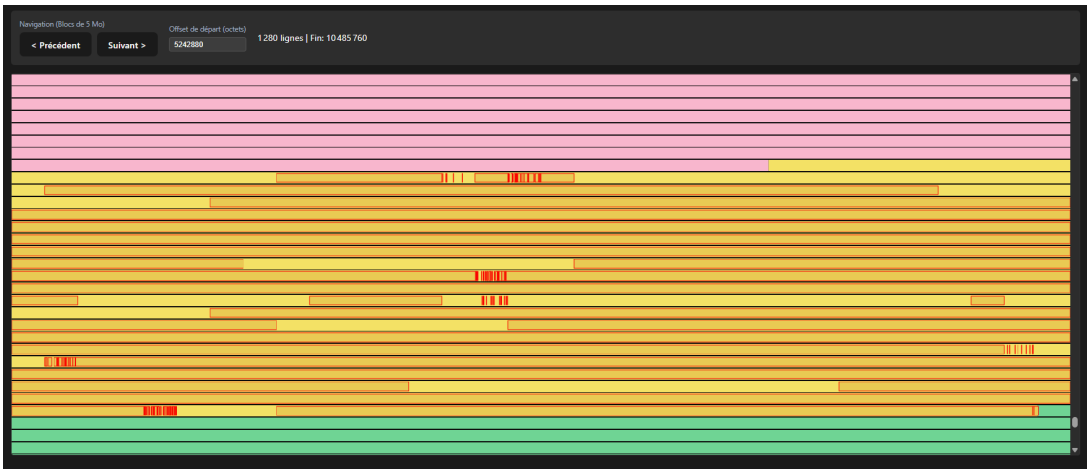

FIGURE 2.11 – Interface de visualisation des erreurs de classification du modèle



FIGURE 2.12 – Interface de visualisation des détails d'une erreur de classification

Sur ces segments, où les erreurs sont surlignées en rouge, il est possible d'avoir un détail sur l'erreur, comme visible dans la Figure 2.12. De plus, en cliquant sur un segment, il est possible d'accéder directement au détail de ce bloc, c'est-à-dire la donnée brute sous forme hexadécimale, montrée dans la Figure 2.13.

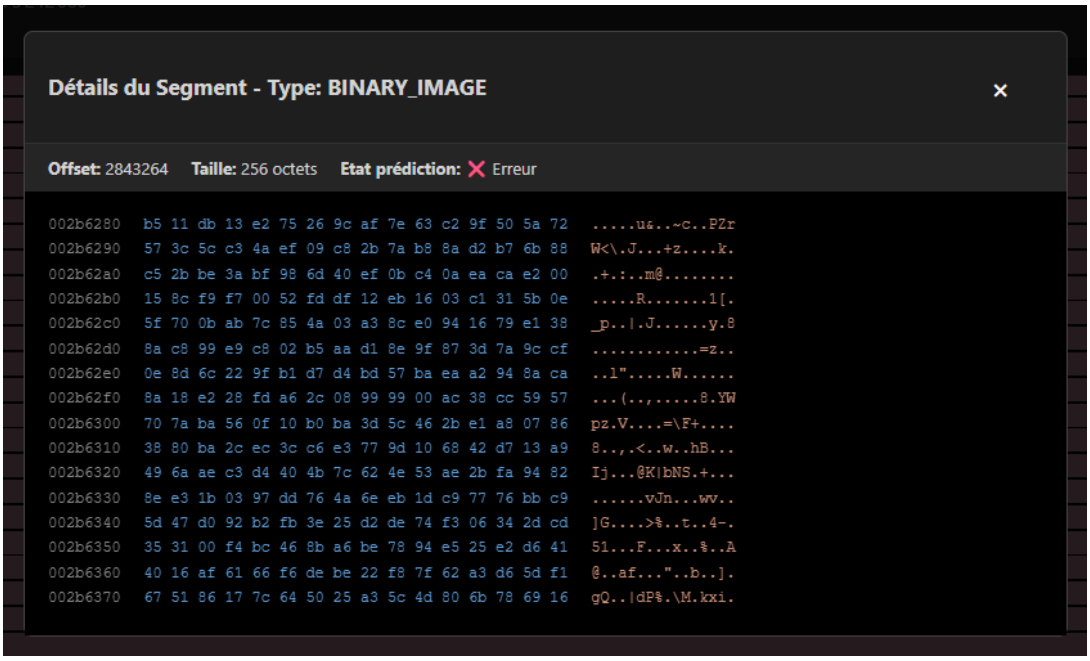

FIGURE 2.13 – Interface de visualisation du détail d'une erreur de classification

C'est l'exploitation de cet outil qui nous a permis d'observer visuellement que les erreurs se concentraient principalement aux frontières des segments, lors des transitions entre deux types de données. Cette découverte a directement motivé l'implémentation du mécanisme de vote pondéré détaillé dans la Sous-section 2.6.2.

3 Résultats

3.1 Le vote pondéré

Accuracy		
Sans vote pondéré	Avec vote pondéré	Diminution de l'erreur
92.77%	94.79%	27.94%

TABLE 3.1 – Performances du vote pondéré

Comme on peut le lire dans le Tableau 3.1, le vote pondéré a permis d'améliorer l'accuracy de 2,02 points, soit une réduction de l'erreur de 27.94%. Cette amélioration significative souligne l'efficacité de cette stratégie pour stabiliser les prédictions du modèle et réduire les artefacts de classification.

3.2 Comparaison des configurations du modèle

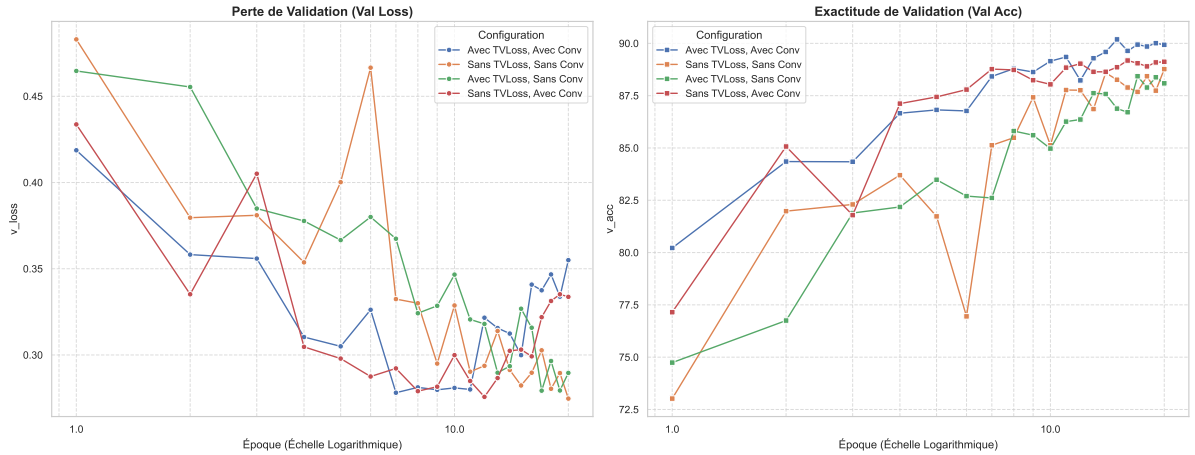

FIGURE 3.1 – Courbes d'apprentissage du modèle (axes des abscisses en échelle logarithmique)

Les résultats ci-dessus, sont obtenus sur de petits jeux de données de 50 Mo, afin de comparer les bénéfices de l'ajout de la *TV Loss* et de la convolution. La configuration avec *TV Loss* et convolution (courbe bleu foncé) se distingue comme étant la plus performante sur l'ensemble de la période d'entraînement. En observant l'évolution de l'exactitude de validation, cette configuration atteint les niveaux les plus élevés, culminant au-delà de 90%

d'exactitude vers la fin des 20 époques. Parallèlement, elle démontre une stabilité remarquable dans la réduction de la perte de validation, se maintenant aux alentours de 0.30 après une phase de stabilisation initiale. À l'inverse, les configurations dépourvues de convolution présentent une plus grande instabilité, avec des fluctuations marquées de la perte et une progression plus lente de l'exactitude, soulignant l'apport crucial de l'architecture convolutive pour la convergence du modèle.

Ici on aurait donc choisi le modèle avec *TV Loss* et Convolution à l'époque 15 qui présente un bon compromis entre l'exactitude et la perte.

3.3 PERFORMANCE GLOBALES

3.3.1 Les performances du modèle avant fine tuning

Matrices de confusion

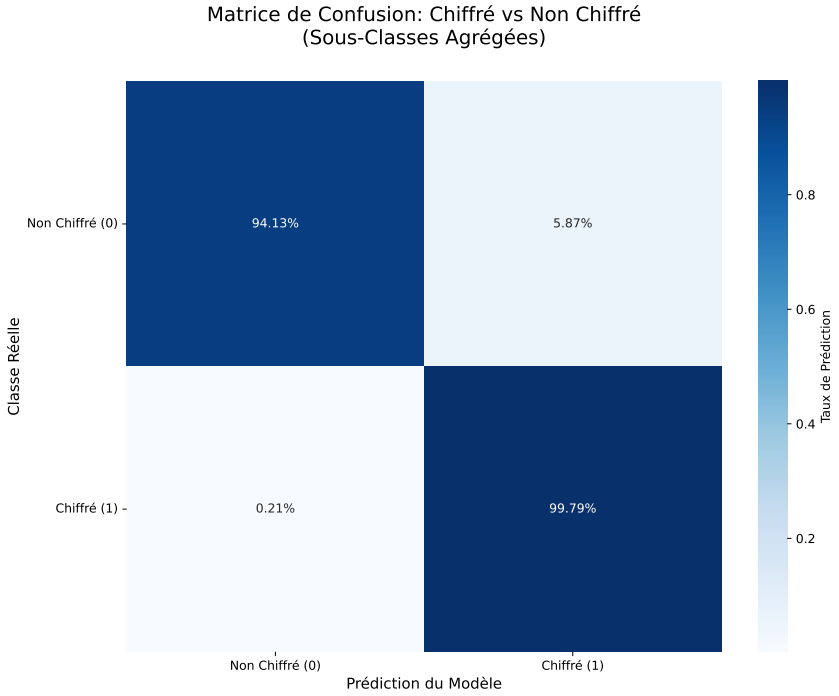

FIGURE 3.2 – Matrice de confusion du modèle

Comme le montre la Figure 3.2, le modèle atteint un taux de vrai positif de 99.79% sur la détection de contenus chiffrés et de 94.13% pour les contenus non chiffrés.

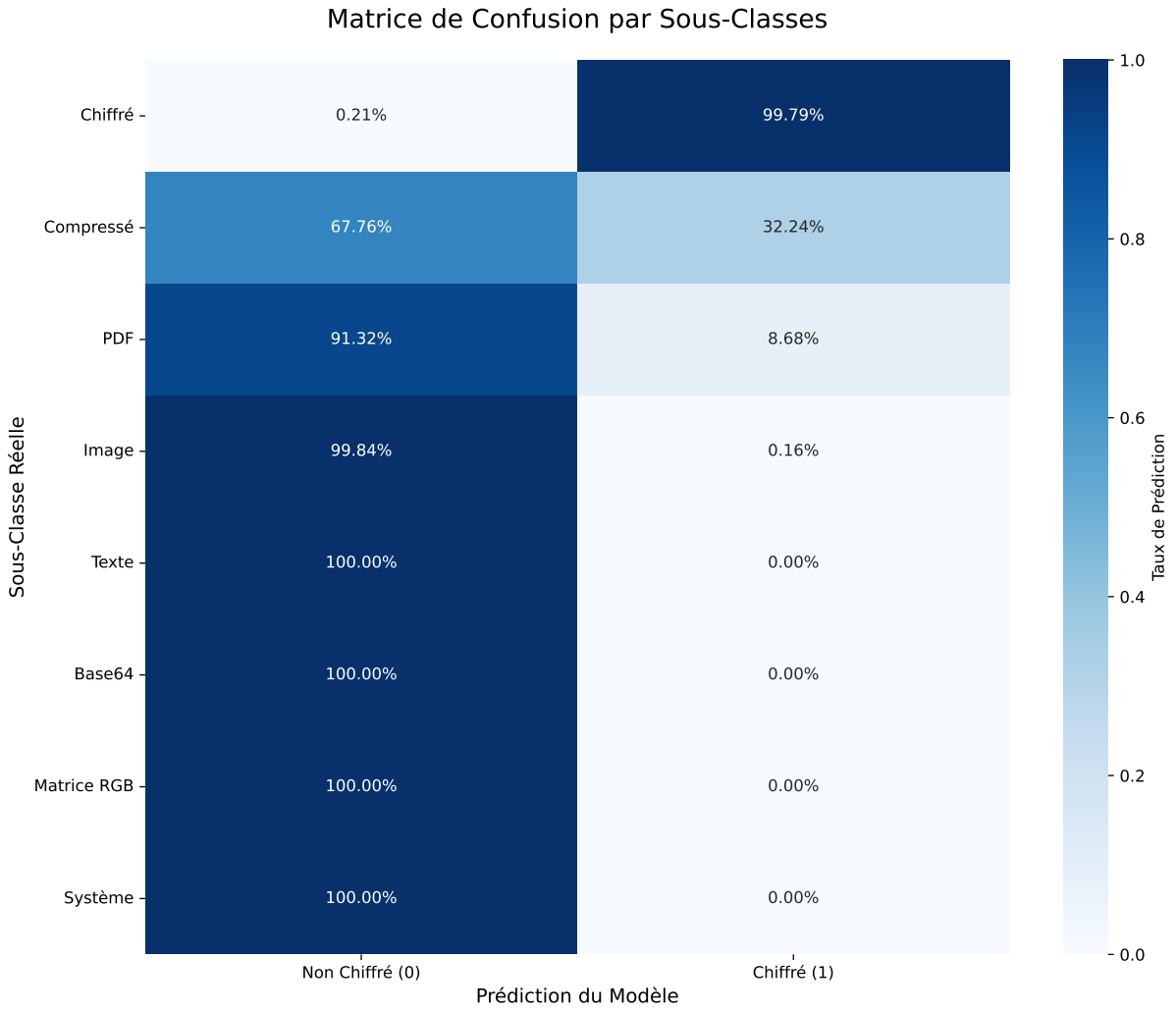

FIGURE 3.3 – Matrice de confusion du modèle, détaillé par type de données

Pour comprendre ces erreurs, nous avons décomposé les résultats sur chaque type de données. On observe que les données compressées sont confondues avec les données chiffrées dans 32.24% des cas, ce qui explique la majorité des erreurs de classification. En effet, les données compressées et chiffrées partagent des caractéristiques similaires, notamment une entropie élevée et une absence de motifs prévisibles, ce qui rend leur différenciation particulièrement difficile pour le modèle.

Métriques d'évaluation

	Score	
Métrique finale	sur le chiffré (0)	sur les non chiffrés (1)
Precision	99.78%	94.44%
Recall	94.13%	99.79%
F1-Score	96.87%	97.05%

TABLE 3.2 – Performances globales du classifieur RAID

Nous observons un déséquilibre entre les performances sur les classes chiffrées et non chiffrées, avec une précision plus élevée sur les données chiffrées mais un rappel plus faible, ce qui confirme que le modèle a du mal à différencier les données compressées des données chiffrées. C'est pourquoi nous avons décidé de procéder à une phase de fine-tuning ciblée pour améliorer la capacité du modèle à distinguer ces deux classes.

3.3.2 Performances du modèle après fine-tuning

Matrices de confusion

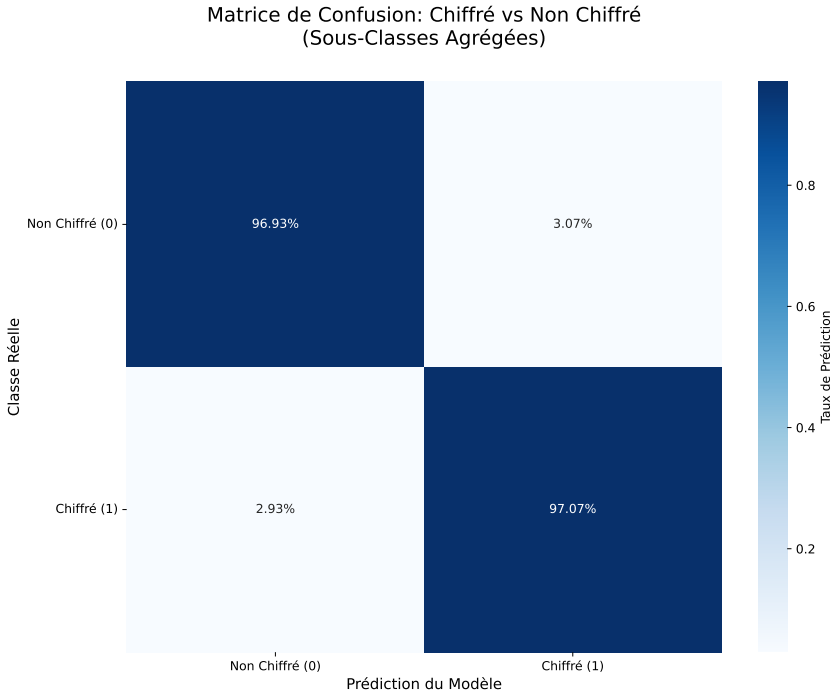

FIGURE 3.4 – Matrice de confusion du modèle

Le modèle atteint un taux de vrai positif de 97.07% sur la détection de contenus chiffrés et de 96.93% pour les contenus non chiffrés. Nous observons une amélioration de l'équilibre entre les classes, avec une réduction des vrais positifs sur les données chiffrées mais une augmentation des vrais positifs sur les données non chiffrées.

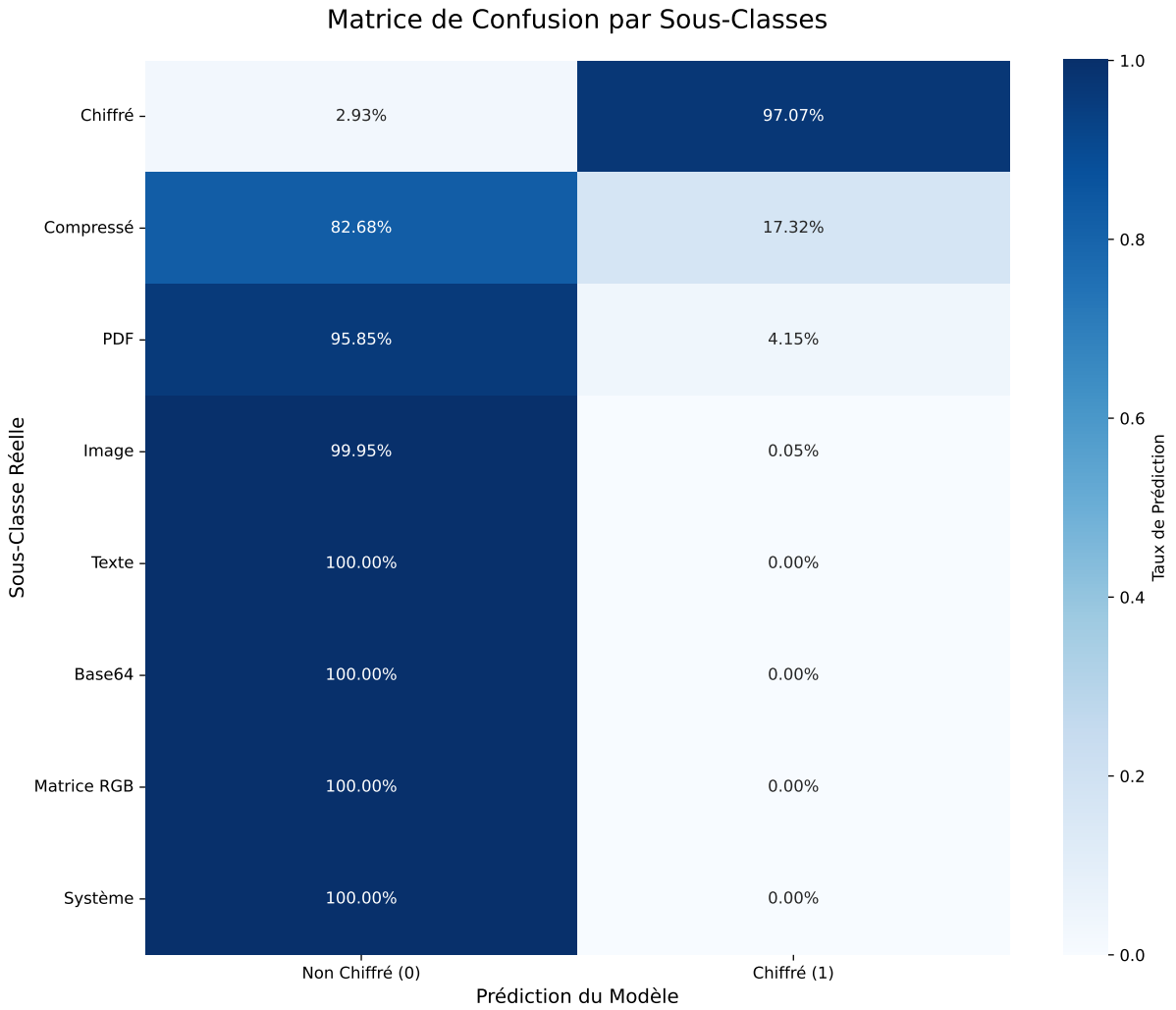

FIGURE 3.5 – Matrice de confusion du modèle, détaillé par type de données

Après fine-tuning, les erreurs sur les données compressées ont été réduites. Nous observons une augmentation du taux d'erreur des données chiffrées.

Métriques d'évaluation

	Score	
Métrique finale	sur le chiffré (0)	sur les non chiffrés (1)
Precision	97.06%	96.94%
Recall	96.93%	97.07%
F1-Score	97.00%	97.00%

TABLE 3.3 – Performances globales du classifieur RAID

Le déséquilibre observé dans le Tableau 3.2 a été corrigé après le fine-tuning, avec une précision et un rappel plus équilibrés entre les classes chiffrées et non chiffrées.

4 Discussion

L'un des défis majeurs de l'analyse de dumps mémoire par un *Transformer* réside dans la volatilité des prédictions locales. Un segment de données peut présenter des caractéristiques ambiguës (par exemple, un en-tête de fichier compressé ressemblant à du texte structuré) qui induisent le modèle en erreur s'il est analysé de manière isolée. L'exploitation de notre outil de visualisation (cf la Section 2.8) a d'ailleurs mis en évidence graphiquement que ces erreurs locales se concentrent principalement aux frontières des segments, lors des transitions physiques entre deux types de données (cf. la Figure 2.10, la Figure 2.11, la Figure 2.12 et la Figure 2.13).

Pour pallier cette limite, nous avons implémenté un mécanisme de vote pondéré afin de stabiliser les prédictions et d'augmenter la fiabilité de la classification des zones de mémoire. En confrontant plusieurs prédictions pour un même octet via des fenêtres glissantes (cf. la Sous-section 2.6.2), nous avons observé une réduction des “artefacts de prédiction" (changements de classe abrupts et incohérents sur quelques octets). Cette approche agit comme un filtre passe-bas, éliminant le bruit haute fréquence dans les résultats pour ne conserver que les zones où le modèle maintient une confiance élevée et constante. Cette réduction des artefacts mène à une augmentation de l'*accuracy* de l'ordre de 2% (cf. le Tableau 3.1).

Il est cependant nécessaire de nuancer cette robustesse par le coût computationnel associé. L'utilisation d'un *stride* multiplie proportionnellement le temps d'inférence. Dans des grands volumes de données, ce compromis entre le temps d'analyse et la précision de la segmentation devient un paramètre critique. Si nos résultats suggèrent que le gain en fiabilité justifie cette latence, elle souligne un arbitrage nécessaire : l'inférence d'un modèle *Transformer* exige des ressources matérielles significatives (GPU) (cf la Section 2.7), là où les approches conventionnelles basées sur des signatures statiques (magic bytes, headers de fichiers ou traces du noyau via des outils comme *Volatility*) restent quasi instantanées et légères. Ces méthodes classiques conservent donc une utilité majeure en l'absence de matériel performant ou lors d'analyses où l'on recherche la rapidité brute.

Toutefois, le projet R.A.I.D. démontre qu'une approche basée sur l'apprentissage profond offre certains résultats que les outils traditionnels pourraient peiner à atteindre. Les méthodes classiques échouent dès lors que les données sont fragmentées ou dépourvues

de leurs en-têtes originaux, une situation pourtant fréquente au sein d'un dump RAM. De plus, les signatures statiques sont particulièrement vulnérables aux techniques d'obfuscation simples, où un changement mineur dans les premiers octets d'un fichier suffit à tromper un scanner. À l'inverse, l'utilisation du mécanisme d'attention (cf. la Section 2.3) permet au *Transformer* de s'affranchir des spécificités du système d'exploitation en cherchant à extraire du sens directement depuis le contenu. En pondérant l'importance de chaque octet dans son contexte global, la classification ne repose plus sur un point de passage unique et rigide, mais sur une compréhension diffuse du bloc de données de 1024 octets (cf. la Section 2.2).

Malgré cette capacité à capturer les motifs et l'organisation profonde des données (la structure d'un texte ou la distribution d'une image), l'analyse des erreurs de classification révèle une limite technique : la confusion récurrente du modèle entre les segments de données chiffrés et compressés (cf. la Section 3.3). Cette difficulté s'explique par la nature mathématique même des algorithmes testés, tels que l'`AES-256-CBC` pour le chiffrement et `zLib` ou `Gzip` pour la compression (cf. la Sous-section 2.1.1). L'objectif d'un algorithme de compression efficace étant d'éliminer toute redondance, il conduit à une distribution des octets proche d'un bruit aléatoire, s'approchant ainsi d'une entropie maximale. Le chiffrement et la compression produisent alors des espaces binaires semblables, dépourvus de motifs répétitifs exploitables par les mécanismes d'attention. De plus, la fragmentation (cf la Sous-section 2.1.3) inhérente à la mémoire vive prive souvent ces segments de leurs métadonnées ou headers spécifiques, empêchant le *Transformer* d'utiliser ces repères fixes pour lever l'ambiguïté.

Dans les résultats d'évaluation, cette confusion entre les deux types se traduit par des faux positifs où des séquences compressées sont classées comme chiffrées. Même avec une validation par vote pondéré, le score de confiance reste élevé pour ces deux catégories : le modèle "est certain" de ne pas faire face à du texte ou des images, mais ne possède pas d'indices suffisants pour distinguer le chiffrement de la compression (cf. la Sous-section 3.3). Cette problématique souligne un défi ouvert de l'apprentissage profond appliqué aux séquences binaires : la distinction entre haute entropie "utile" (compression) et "intentionnelle" (chiffrement).

Lors des premières évaluations, cette confusion entre les deux types se traduit par des faux positifs au point qu'environ 30% des données compressées étaient faussement étique-

tées comme chiffrées (cf la Figure 3.3). Même avec une validation par vote pondéré, le score de confiance restait élevé pour ces deux catégories, le modèle étant "certain" d'exclure les structures types textes ou images, mais ne possédant pas d'indices suffisants pour distinguer le chiffrement de la compression. Cette problématique souligne un défi ouvert de l'apprentissage profond appliqué aux séquences binaires : la distinction entre haute entropie "utile" (compression) et "intentionnelle" (chiffrement).

Pour dépasser ce blocage, nous avons procédé à une phase de fine-tuning ciblé. En réentraînant notre meilleur modèle sur trois époques avec un jeu de données strictement équilibré (répartition 50/50 entre données chiffrées et compressées), nous avons significativement accru sa capacité de discernement. Cela a permis d'obtenir :

— Une réduction de 27% des erreurs de classification. (cf la Figure 3.5).
— Un gain de près de 2% d'accuracy globale, validant l'efficacité de la spécialisation du modèle. (cf la Figure 3.4)

Ces résultats ne sont pas issus d'un gain pur mais plutôt d'un rééquilibrage, nos données présentaient un déséquilibre de la précision avant le fine tuning, avec 99.78% sur le chiffré et 94.44% sur le non chiffré (cf le Tableau 3.2). Après la passe de fine-tuning, cette précision s'équilibre avec 97.06%de précision pour le chiffré et 96.94% pour le non chiffré (cf le Tableau 3.3). Malgré la diminution de la précision sur le chiffré, le gain se montre considérable sur l'ensemble.

Pour dépasser cette limite, l'intégration d'une surcouche via l'extraction de features statistiques, comme un test du Khi-deux (χ^2) ou une extension du contexte, constituerait une piste prometteuse. Nous ne sommes pas capables de prouver l'efficacité de ces deux approches par manque de ressources, mais nous les voyons comme de probables candidats à l'amélioration de cette distinction.

Au-delà de ces limites techniques, et bien que les résultats soient prometteurs, il est important de souligner que le modèle a été entraîné et évalué sur un jeu de données synthétique. Ce dernier s'efforce de reproduire la diversité d'un dump mémoire, mais il existe inévitablement de grands écarts avec la complexité d'une mémoire vive réelle, notamment au niveau de la fragmentation dynamique (cf la Sous-section 2.1.3) et de la présence de formats ou de structures de données propriétaires liés aux systèmes d'exploitation. Si ces formats paraissent trop éloignés des motifs appris sur le corpus d'entraînement, leur non-exhaustivité influencera négativement les performances du classifieur. Cette dépendance à

un dataset synthétique pose directement la question de la validité externe de nos résultats : les performances globales observées ne peuvent très probablement pas être calquées sur des scénarios réels dans des dumps de RAM véridiques, cela nécessiterait une phase de validation et de réentrainement au besoin.

Ces limites ouvrent néanmoins des perspectives majeures, à commencer par le passage à un apprentissage non supervisé. Actuellement, le système repose sur un entraînement supervisé à l'aide d'un générateur de datasets qui définit arbitrairement les classes de données. Cependant, la réalité d'un dump RAM système est bien plus complexe et hétérogène que les échantillons produits synthétiquement, un mode non supervisé permettrait au modèle d'apprendre directement les structures statistiques intrinsèques de la RAM sans nécessiter d'étiquetage préalable, facilitant ainsi la détection d'artefacts encore inconnus.

L'architecture *Transformer* de R.A.I.D. (cf la Section 2.5) est particulièrement adaptée à cette transition via des techniques de pré-entraînement similaires à celles utilisées dans les modèles de langage (type BERT). En masquant certains octets au sein d'une séquence et en demandant au modèle de les prédire (architecture decoder), le système développerait une compréhension profonde de la "syntaxe" binaire des différents types de données. Dans ce cadre, une erreur de prédiction élevée deviendrait un indicateur natif d'une zone à haute entropie (chiffrement), sans que le modèle ait besoin d'avoir "vu" des données chiffrées durant son entraînement.

L'objectif final de cette perspective est de permettre au modèle de s'adapter dynamiquement à n'importe quel système d'exploitation ou architecture matérielle. En passant au non supervisé, R.A.I.D. ne chercherait plus à valider des catégories pré-définies, mais à segmenter la mémoire en zones de cohérence. Cette approche permettrait d'isoler avec une plus grande précision les zones de données volatiles, posant ainsi les bases d'une assistance automatisée capable de s'auto-ajuster à la spécificité de chaque dump RAM analysé, bien que cela reste hypothétique.

Enfin, le projet R.A.I.D. a validé la capacité des Transformers à classifier des segments isolés, une étape suivante pourrait consister en la gestion de la fragmentation et la reconstruction logicielle des fichiers, un dump RAM n'étant pas une suite linéaire de fichiers, mais un agglomérat de blocs mémoires souvent éparpillés.

Le modèle actuel travaille sur des fenêtres de 1024 octets (chunk_size) (cf. la Section 2.2), ce qui lui permet d'identifier avec précision la nature d'un fragment (image, PDF, texte). L'étape suivante impose de déterminer si deux segments identifiés comme "IMAGE", mais situés à des adresses distantes, appartiennent au même objet binaire. L'utilisation du mécanisme d'attention pourrait ainsi être étendue pour comparer les similitudes entre différents blocs afin de détecter des continuités logiques malgré la fragmentation physique. En se positionnant comme une brique de base pour les outils de *file carving* intelligents, R.A.I.D. offrirait une perspective de "carving sémantique" face aux outils traditionnels (*Scalpel*, *Foremost*) limités par la disparition des en-têtes en RAM :

— D'une part, en isolant les zones d'intérêt au milieu du bruit ou des données chiffrées, le modèle réduirait drastiquement l'espace de recherche pour les algorithmes de reconstruction.

— D'autre part, il permettrait une reconstruction sans en-tête en reconnaissant des structures (comme des pixels bruts) même lorsque les métadonnées originales ont été effacées, rendant possible la récupération de documents partiels là où les méthodes classiques échouent.

À terme, l'intégration de R.A.I.D. dans une chaîne de traitement automatisée permettrait de proposer une prévisualisation des fichiers fragmentés. En couplant la classification par octets avec des techniques de réassemblage basées sur la probabilité de transition entre les types de données, le projet pose les bases d'un outil capable de reconstituer des preuves volatiles (fragments de conversations, images de sessions) qui sont aujourd'hui souvent considérées comme perdues car trop déstructurées.

5 Conclusion

Le projet R.A.I.D. (RAM Artificial Inspection Dump) a permis de démontrer l'efficacité des architectures *Transformer* pour l'analyse automatisée de la mémoire vive, un domaine où la volumétrie et la fragmentation des données sont potentiellement un obstacle pour les outils plus traditionnels. En nous affranchissant des signatures statiques au profit d'une approche basée sur la compréhension statistique des flux binaires, nous avons développé un outil capable de segmenter un dump mémoire avec une précision de 94.79%.

L'implémentation du modèle `BytesTransformerClassifier`, couplée à une couche de convolution locale et à un mécanisme de self-attention, a prouvé sa capacité à identifier la nature profonde des données (texte, images, documents) même en l'absence de métadonnées ou d'en-têtes explicites. L'introduction du processus de vote pondéré a constitué une étape clé de notre méthodologie, permettant de stabiliser les prédictions et de réduire fictivement le bruit de classification inhérent à l'analyse par fenêtres glissantes.

Ces résultats apportent une première réponse positive à notre problématique : les méthodes profondes s'avèrent pleinement fonctionnelles et viables pour l'interprétation sémantique du binaire brut, et ce, malgré l'absence d'une comparaison exhaustive avec l'ensemble des approches traditionnelles. Le modèle `BytesTransformerClassifier` a prouvé sa capacité à extraire des structures ordonnées de manière autonome.

Toutefois, ce projet a également mis en lumière des défis persistants, notamment la convergence statistique entre les données chiffrées et compressées. Cette limite souligne que si l'apprentissage profond excelle dans la reconnaissance de structures ordonnées, la distinction entre différentes formes de haute entropie nécessite encore l'intégration de métriques mathématiques plus fines, de contextes d'analyse élargis ou d'opérations de fine-tuning.

En conclusion, R.A.I.D. offre une nouvelle base d'outils de forensique assistée par intelligence artificielle. Les perspectives ouvertes par ce travail, qu'il s'agisse du passage à l'apprentissage non supervisé pour s'adapter dynamiquement de nouveaux systèmes, ou du développement d'un moteur de file carving sémantique, suggèrent que l'IA peut transformer radicalement la manière dont les enquêteurs extraient des preuves volatiles d'un système. Ce projet démontre que, face à la complexité croissante des environnements numériques,

la capacité des modèles de langage à "lire" le binaire brut constitue un atout majeur pour l’informatique légale de demain.

Répartition du travail

Il est complexe sur ce projet d'expliciter la portion de code produite par chacun, le projet ayant été conçu petit à petit, les ajouts étaient épars et réalisés après réflexion commune sur la machine la plus puissante au moment de l'ajout, chacun a participé également à toutes les parties (code, rapport, réflexion ...). Les points ayant eu plus d'emphase par une personne sont explicités ici :

— Amand -> Transformer (TVLoss, Scheduler, convolutions, BF16 ...)
— Théo -> Optimisation GPU & CPU, Visualiseur, dataSetGenerator
— Tom -> vote avec pondération, structure de base du transformer, rapport LateX

Bibliographie

GOUGEON, Thomas. « Analyse forensique de la mémoire des cartes à puce ». 2017, theses.hal.science/tel-01661305/document.

VASWANI, Ashish, et al. « Attention is All you Need ». *Advances in Neural Information Processing Systems (NIPS)*. 2017, pages 5998-6008, arxiv.org/abs/1706.03762.