Apport de l'augmentation de données au décodage neuronal souple par syndrome des codes correcteurs d'erreurs

Raphaël LE BIDAN Ahmad ISMAIL Elsa DUPRAZ Charbel ABDEL NOUR

Lab-STICC UMR CNRS 6285
IMT Atlantique, Dépt MEE, Technopôle Brest-Iroise, CS 83818, 29238 Brest Cedex 03

Résumé – Nous montrons comment tirer profit des automorphismes d'un code pour accroître la capacité d'un décodeur neuronal à apprendre, généraliser, et renforcer la qualité de ses prédictions, par de l'augmentation de données à l'entraînement et à l'inférence.

Abstract – We show how to leverage the automorphisms of an error-correcting code to enhance a neural decoder's ability to learn and generalize, as well as to improve its prediction quality through data augmentation during training and inference.

1 Introduction

On s'intéresse ici au problème du décodage souple d'un code en bloc linéaire quelconque. Le décodage par statistique d'ordre (*Ordered Statistics Decoding* ou OSD) [4] est une solution possible. Le décodage par propagation de croyance (*Belief Propagation* ou BP) en est une autre [9]. A ce jour, on ne connaît toutefois pas de solution qui atteigne un compromis satisfaisant entre performance de correction et complexité matérielle. A ce titre, le décodage par apprentissage profond est une piste qu'il convient d'explorer, pour en évaluer le potentiel, les limites, et peut-être aussi, qui sait, découvrir de nouveaux algorithmes par cette voie. Le décodage neuronal par syndrome proposé dans [1] est une approche particulièrement intéressante. Elle n'est pas encore en mesure de rivaliser avec les solutions traditionnelles évoquées ci-dessus. Mais c'est une piste prometteuse, car il est possible de l'améliorer à de nombreux niveaux, par exemple en s'inspirant des bonnes pratiques ayant fait leurs preuves en apprentissage profond. Cet article s'inscrit précisément dans cet objectif. Après avoir expliqué comment s'appuyer sur la théorie des codes pour faire de l'augmentation de données dans le contexte du décodage neuronal, nous montrons ensuite comment tirer profit de cette augmentation pour non seulement accroître la capacité d'un décodeur neuronal par syndrome à apprendre et généraliser durant l'entraînement, mais aussi renforcer la qualité de ses prédictions à l'inférence.

2 Décodage neuronal par syndrome

2.1 Principe

Soit un code en bloc linéaire $\mathcal{C}$ de longueur n et dimension k, et $\mathbf{H}$ une matrice de contrôle de parité pour $\mathcal{C}$. Par la suite, on désigne par $\mathbb{F}_2^n$ l'espace vectoriel de dimension n construit sur le corps de Galois binaire $\mathbb{F}_2 = \{0, 1\}$. Soit $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{R}^n$ l'observée bruitée relative à la transmission d'un mot $\mathbf{c} = (c_1, \ldots, c_n)$ de $\mathcal{C}$ à travers un canal à bruit additif blanc gaussien, au moyen d'une modulation de phase à deux états. On s'intéresse au décodage souple de l'observée $\mathbf{y}$. Dans ce contexte, il est bien connu que le décodeur à Maximum de Vraisemblance (*Maximum-Likelihood Decoder*, ou MLD) minimise la probabilité de se tromper sur le mot de code émis, et que la règle de décodage MLD peut prendre différentes formes. Celle qui nous intéresse dans le cas présent consiste à prendre une décision ferme $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbb{F}_2^n$ sur l'observée $\mathbf{y}$ par seuillage de ses composantes, calculer le syndrome $\mathbf{s} = \mathbf{z}\mathbf{H}^t \in \mathbb{F}_2^{n-k}$, puis rechercher le motif d'erreur binaire $\mathbf{e}_{\text{ML}} \in \mathbb{F}_2^n$ qui minimise la fonction de coût $w(\mathbf{e}) = \sum_{i:e_i=1} |y_i|$ (motif d'erreur *le plus vraisemblable*), parmi les 2^k motifs candidats composant la classe latérale associée au syndrome $\mathbf{s}$ [10]. La décision sur le mot de code émis s'en déduit par correction de l'erreur estimée : $\hat{\mathbf{c}} = \mathbf{z} - \mathbf{e}_{\text{ML}}$. Il s'ensuit que la paire $(|\mathbf{y}|, \mathbf{s})$ constitue une statistique suffisante pour le décodage MLD. Pour la plupart des codes en bloc, la recherche du motif d'erreur le plus vraisemblable est un problème difficile. Contrairement au décodage ferme, ce motif ne peut être tabulé à l'avance pour chaque valeur du syndrome car la fonction de coût à minimiser dépend de l'observée $\mathbf{y}$. Comme proposé dans [1], l'idée est donc d'utiliser un réseau de neurones pour approcher le décodage MLD et tâcher d'estimer, à moindre coût, le motif d'erreur $\mathbf{e}_{\text{ML}}$ à partir de la donnée du couple $(|\mathbf{y}|, \mathbf{s})$. La figure 1 résume le principe général du décodage souple neuronal par syndrome. Parmi les questions qui se posent alors : jusqu'à quel point est-il possible d'approcher le décodage MLD avec un réseau de neurones ? Quel type de modèle utiliser à cet effet ? Avec combien de paramètres, et quel jeu d'entraînement (taille, distribution) ?

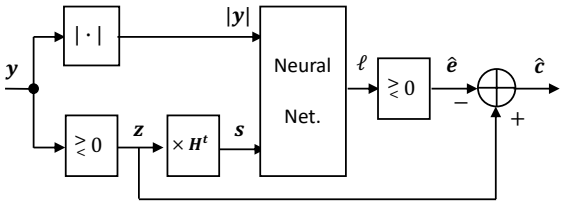

FIGURE 1 : Principe du décodage neuronal par syndrome.

2.2 Choix du modèle

Pour cet article, nous avons choisi de nous limiter à un modèle de décodeur neuronal récurrent de type *Gated-Recurrent Unit*

(GRU). En effet, d'autres auteurs ont remarqué [1, 3], et nous l'avons constaté aussi, qu'en l'état actuel des connaissances, ce modèle affiche des performances au moins aussi bonnes que d'autres architectures telles que le perceptron multi-couches ou bien le *Transformer* [2]. Ceci dit, tous les principes que nous allons présenter sont indépendants du modèle choisi.

Le décodeur neuronal GRU par syndrome introduit dans [1] se compose de L cellules GRU élémentaires empilées les unes sur les autres. Chacune met à jour un vecteur latent (état dit *caché*) de dimension h. L'ensemble itère T fois sur la répétition de la même entrée $(|\mathbf{y}|, \mathbf{s}) \in \mathbb{R}^{2n-k}$ formée de la concaténation du vecteur des fiabilités bits et du syndrome, ce dernier étant exprimé sous forme bipolaire ± 1. Une couche linéaire de taille $h \times n$ est placée à la sortie de l'architecture GRU empilée pour extraire de la représentation latente calculée par le réseau sa prédiction $\boldsymbol{\ell} \in \mathbb{R}^n$ sur le motif d'erreur recherché. $\boldsymbol{\ell}$ est un vecteur de *logits*, dont les composantes peut être vues comme une estimée du logarithme de rapport de vraisemblance *a posteriori* sur chaque bit codé. Le signe de ℓ_i donne la décision ferme $\hat{e}_i$ sur chaque bit du motif d'erreur.

Le nombre de paramètres du décodeur GRU augmente en $O(Lh^2)$. On ne connaît pas encore de recette ou loi d'échelle pour choisir les paramètres L et T. $L = 5$ couches et $T = 3$ à 5 itérations sont des valeurs typiques dans la littérature. Par contre, il est suggéré dans [1] de choisir la taille d'état caché h égale à 5 ou 6 fois la dimension $2n - k$ du vecteur d'entrée.

2.3 Recette d'entraînement

Dans le cadre de cette étude, l'entraînement du décodeur neuronal par syndrome se fait de manière supervisée, à partir d'un jeu d'entraînement $\{(|\mathbf{y}^{(i)}|, \mathbf{s}^{(i)}); \mathbf{e}_{\text{ML}}^{(i)}\}_{i=[\![1,M]\!]}$ *construit au préalable*, et dont il est possible d'optimiser la distribution [7]. On se limitera ici à des jeux d'entraînements construits par « simple » simulation de Monte-Carlo utilisant un décodeur OSD d'ordre suffisant pour garantir que les motifs d'erreurs $\mathbf{e}_{\text{ML}}^{(i)}$ collectés à la sortie de ce décodeur correspondent bien aux décisions du décodeur MLD. Nos expériences ont montré que, pour la plupart des codes, entraîner sur un seul point de rapport signal-à-bruit (RSB) mais bien choisi généralise très bien sur une large plage autour de cette valeur cible. Il convient de préciser par ailleurs que le décodeur n'est entraîné que sur des exemples dont le syndrome est non nul.

Le décodage souple d'un code correcteur est un problème de classification multi-classes. En principe, un décodeur neuronal devrait donc être entraîné à minimiser l'entropie croisée, mais le nombre prohibitif de classes, 2^k, interdit cette approche. Une solution consiste à décomposer le décodage du mot reçu en n problèmes de classification binaire, et à minimiser l'entropie croisée binaire moyenne résultante :

$$\mathcal{L}(\boldsymbol{\ell}, \mathbf{e}) = -\frac{1}{n} \sum_{i=1}^{n} [e_i \log(\sigma(\ell_i)) + (1 - e_i) \log(1 - \sigma(\ell_i))]$$

$\sigma(x)$ désignant la fonction sigmoïde. Le prix à payer pour ce découplage du problème d'optimisation initial est un décodeur qui tend à s'écarter un peu de l'objectif de prédire correctement le motif d'erreur dans son ensemble, pour mieux classifier chaque bit pris individuellement. On observe que les modèles entraînés de la sorte affichent généralement un très bon taux d'erreur binaire après décodage, proche de celui du décodeur MLD, alors que leur performance en terme de taux d'erreur par mot est plus en retrait. [3] a remarqué très judicieusement qu'estimer les n bits du motif d'erreur affectant le mot de code pouvait se ramener au problème équivalent, un peu plus simple pour le modèle, d'estimer seulement les k bits du motif d'erreur entachant le message. Cela peut rendre l'apprentissage d'autant plus aisé pour le modèle que la dimension k du code est petite devant n, et contribuer ainsi à en améliorer la précision. Dans le cadre de l'étude présentée ici, nous avons souhaité rester au plus près de l'approche originelle de [1], mais présenterons néanmoins un résultat obtenu avec [3].

3 Augmentation de données par automorphismes pour l'entraînement

Nous avons montré dans [7] que, pour qu'un décodeur neuronal approche au plus près la performance du décodeur MLD, il est nécessaire de l'entraîner sur des exemples générés par... un décodeur MLD (ou assimilé). Construire le jeu d'entraînement peut s'avérer très coûteux en temps de calcul, dès lors que l'on s'attaque à des codes puissants, avec une distance minimale d élevée, même avec un décodeur simplifié tel que l'OSD. On voudra donc entraîner notre décodeur sur le plus petit nombre d'exemples nécessaire, pour minimiser le coût de la construction du jeu de données, mais aussi économiser l'espace disque requis pour le stocker. S'agissant d'entraîner au mieux un modèle à partir d'un nombre limité d'exemples, l'augmentation de données est une pratique courante en apprentissage profond. Elle consiste, pour chaque exemple de chaque lot (*batch*) présenté au modèle durant l'entraînement, à lui appliquer une transformation tirée au hasard dans un ensemble prédéfini. Ce faisant, on expose le modèle à une plus grande diversité d'exemples, ce qui renforce sa capacité à apprendre et généraliser. Nous proposons ici d'utiliser les automorphismes du code à cet effet.

3.1 Groupe d'automorphismes d'un code

Soit π une permutation de l'ensemble des entiers $[\![1, n]\!]$ et $\pi(\mathbf{c})$ le vecteur résultant de l'application de cette permutation aux coordonnées du vecteur $\mathbf{c}$. Une permutation de coordonnées envoyant chacun des mots du code $\mathcal{C}$ vers un autre mot de $\mathcal{C}$, pas nécessairement distinct du mot de départ, définit un *automorphisme* de $\mathcal{C}$. L'ensemble de ces automorphismes forme un groupe noté $\text{Aut}(\mathcal{C}) = \{\pi : \pi(\mathbf{c}) \in \mathcal{C}\ \forall \mathbf{c} \in \mathcal{C}\}$. Pour certains codes, ce groupe peut se réduire à la permutation identité. On ne connaît complètement $\text{Aut}(\mathcal{C})$ que pour un très petit nombre de codes. Il existe des algorithmes, tel que l'algorithme de Léon, permettant de calculer ce groupe pourvu que le code ne soit pas trop long. Les codes pris en exemple dans cet article sont un code BCH binaire primitif et un code LDPC quasi-cyclique. On sait [8] que le groupe d'automorphismes d'un code BCH binaire primitif de longueur $n = 2^m - 1$ inclut le sous-groupe des décalages cycliques $\{\pi_s : i \mapsto (i + s) \bmod n, i = [\![1, n]\!]\}_{s=[\![0,n-1]\!]}$, ainsi que le sous-groupe des permutations de Frobenius $\{\pi_j : i \mapsto i2^j \bmod n, i = [\![1, n]\!]\}_{j=[\![0,m-1]\!]}$, soit au moins $m \times n$ permutations par composition des deux. Le groupe d'automorphisme d'un code quasi-cyclique construit à partir de matrices circulantes de taille Z est souvent beaucoup plus restreint, mais comprend *a minima* les Z permutations quasi-cycliques

$\{\pi_q : i \mapsto \lfloor i/Z \rfloor Z + (i+q) \bmod Z, i = [\![1, n]\!]\}_{q=[\![0,Z-1]\!]}$.

3.2 Application des automorphismes à l'augmentation du jeu d'entraînement

On vérifie aisément que si $\mathbf{e}_{\text{ML}}$ est le motif d'erreur MLD correspondant à la donnée de $(|\mathbf{y}|, \mathbf{s})$, alors, pour toute permutation $\pi \in \text{Aut}(\mathcal{C})$, $\pi(\mathbf{e}_{\text{ML}})$ est la décision MLD pour l'observée transformée $(|\pi(\mathbf{y})|, \mathbf{s}' = \pi(\mathbf{z})\mathbf{H}^t)$. Ainsi, moyennant une permutation du vecteur des fiabilités $|\mathbf{y}|$ et du motif d'erreur cible $\mathbf{e}_{\text{ML}}$, et le recalcul à la volée du syndrome associé, l'utilisation des automorphismes du code offre un moyen simple pour introduire de la diversité dans les exemples présentés au décodeur neuronal au fur et à mesure de son entraînement.

Afin d'évaluer la pertinence de cette approche, nous avons entraîné un modèle GRU $L = 5, T = 3, h = 246$ (1.7 millions de paramètres) à décoder le code BCH $(31, 21, 5)$. Dans un premier temps, le modèle a été entraîné sur 3 jeux d'entraînements composés respectivement de 1, 4, et 16 millions d'exemples *tous distincts*. Les données ont été générées au moyen d'un décodeur OSD d'ordre 2, à un RSB cible $E_b/N_0 = 3$ dB, E_b désignant l'énergie dépensée par bit d'information transmis, et $N_0/2$ la densité spectrale de puissance bilatérale du bruit. Tous les entraînements ont été menés sur des batchs de 4096 exemples et un total de 128 époques. Nous avons ensuite reconduit l'expérience avec deux jeux de données augmentés, formés de 4 et 16 millions d'exemples. Tous deux ont été construit à partir du jeu de taille 1 million, en appliquant à chaque exemple d'abord 4 puis 16 permutations tirées au hasard parmi les $5 \times 31 = 155$ permutations à disposition pour ce code. Cette expérience est un pire cas de diversité puisque l'augmentation a été réalisée ici, volontairement, *à la construction du jeu de données* et non à la volée. Les exemples permutés ont été figés une fois pour toute et revus à l'identique par le modèle à chaque époque de l'entraînement, au lieu d'en regénérer de nouveaux au fil de l'eau. Malgré cela, on peut voir sur les performances des différents modèles rapportées sur la figure 2 que cela ne fait aucune différence avec le cas idéal d'exemples tous distincts, ce qui est un résultat remarquable.

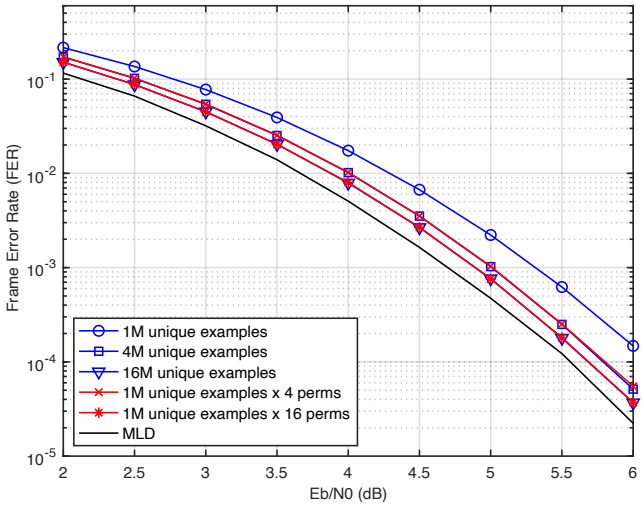

FIGURE 2 : Taux d'erreur mot pour un décodeur GRU entraîné sur le code BCH (31,21,5) avec des jeux de motifs d'erreurs ML uniques ou bien augmentés par automorphismes.

4 Augmentation de données par automorphismes pour l'inférence

L'intérêt de l'augmentation de données ne s'arrête pas à l'entraînement. On peut l'exploiter aussi pour améliorer la précision des prédictions du modèle en phase d'inférence, une stratégie appelée *Test-Time Augmentation* (TTA).

4.1 Principe de l'approche TTA

Chaque observée fournie au modèle subit une série de transformations. La prédiction finale sur l'observée s'obtient par agrégation des prédictions faites par le modèle sur chaque transformée. On lui donne ainsi plus de chances de voir quelque chose qu'il a appris à reconnaître, et donc de calculer la bonne prédiction. Le principal inconvénient est un coût d'inférence plus élevé, proportionnel au nombre de transformations utilisées. Transposé au décodage des codes correcteurs, et avec une transformation de type automorphisme du code, on retrouve une stratégie bien connue en codage, le *décodage par permutations*, dont l'origine remonte aux travaux de Prange dans les années 60 et qui connaît actuellement un regain d'intérêt dans le cadre du décodage souple des codes courts [5].

4.2 Application à l'amélioration des prédictions du décodage neuronal par syndrome

Partant d'une observée $(|\mathbf{y}|, \mathbf{s})$, l'augmentation de données par automorphismes à l'inférence consiste à lui appliquer P permutations distinctes π_j prises au hasard dans $\text{Aut}(\mathcal{C})$. Pour chaque transformée $(|\pi_j(\mathbf{y})|, \pi_j(\mathbf{z})\mathbf{H}^t)$ de cette observée, on calcule la prédiction permutée $\pi_j(\boldsymbol{\ell}_j)$ à qui on applique ensuite la permutation réciproque π_j^{-1} pour récupérer $\boldsymbol{\ell}_j$. Si une ou plusieurs prédictions donnent des mots de code, on retourne le candidat le plus vraisemblable. En l'absence de candidat, cas le plus fréquent, nous avons évalué deux manières de combiner les prédictions pour former la prédiction finale $\boldsymbol{\ell}$: moyenne des logits sur chaque coordonnée, ou bien sélection de la prédiction la plus confiante (logit de valeur absolue maximale) en chaque bit. Prendre la moyenne empirique $\boldsymbol{\ell} = \frac{1}{P}\sum_{j=1}^{P} \boldsymbol{\ell}_j$ des différentes prédictions s'est avéré la meilleure option.

Nous avons tout d'abord appliqué cette stratégie à notre modèle GRU de 1.7 millions de paramètres entraîné à décoder le code BCH $(31, 21, 5)$ sur un jeu d'entraînement fixe de 4 millions d'exemples tous distincts, collectés à 3 dB. A chaque observée générée pour évaluer la performance du modèle, nous avons appliqué successivement 4 puis 16 permutations prises au hasard dans les 155 automorphismes à disposition, réalisé l'inférence comme expliqué ci-dessus, et mesuré le taux d'erreur par mot sur le résultat. La figure 3 présente les performances obtenues. En comparant avec la figure 2, on remarque que 4 permutations suffisent à égaler la performance du même modèle mais entraîné sur 4 fois plus de données (16 millions d'exemples). On voit aussi qu'il n'y pas d'intérêt à considérer davantage de permutations dans ce cas.

Nous avons reconduit l'expérience mais sur un code plus long et plus puissant, le code LDPC irrégulier quasi-cyclique $(96, 48, 10)$ tiré de [6] et construit sur des circulantes de taille $Z = 8$. Nous avons entraîné un modèle GRU de 39 millions

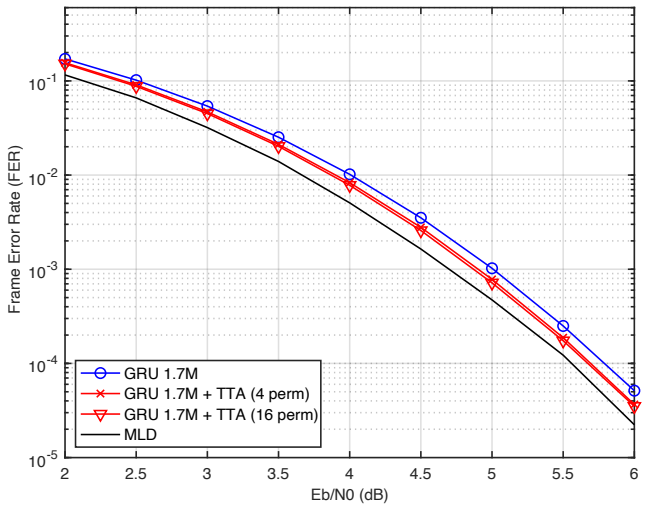

FIGURE 3 : Taux d'erreur mot mesuré sans et avec augmentation de données à l'inférence (TTA) sur le code BCH (31,21,5).

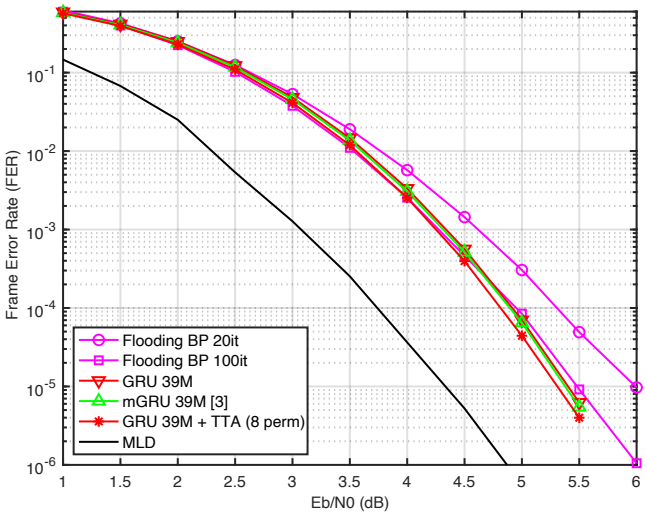

FIGURE 4 : Taux d'erreur mot mesuré sans et avec augmentation de données à l'inférence (TTA) sur le code LDPC quasi-cyclique (96,48,10), et comparaison avec le décodeur BP.

de paramètres à le décoder, sur un jeu de $2^{26} = 67$ millions d'exemples collectés à 3 dB au moyen d'un décodeur OSD d'ordre 3. L'entraînement a été mené sur des batchs de 4096 exemples, durant 128 époques, en appliquant à la volée la technique d'augmentation de données de la section 3 avec les 8 permutations quasi-cycliques du code. La figure 4 présente la performance mesurée sans et avec augmentation de données à l'inférence. Nous y avons également rapporté, en référence, la performance du décodeur BP à 20 et 100 itérations, ainsi que celle du décodeur GRU orienté message (mGRU) proposé dans [3], qui, sur cet exemple, s'avère faire à peine mieux que le décodeur GRU d'origine, orienté mot. On remarque que BP et GRU ont une performance comparable à faible RSB. Ensuite le décodeur GRU surpasse progressivement le BP au fur et à mesure que le RSB augmente. On note aussi que l'augmentation de données à l'inférence apporte un gain de 0.1 dB. Pour autant il reste encore 1 dB à combler par rapport au décodeur MLD. Cela peut paraître beaucoup au regard du coût de calcul de ce modèle GRU, très largement supérieur à celui du décodeur BP. Un rapide calcul révèle toutefois que ce modèle est sous-entraîné dans le sens où il n'a pu voir qu'une infime fraction, 0.0002% au mieux, des valeurs possibles du syndrome à l'entraînement. Vu sous cet angle, la performance de la figure 4 nous paraît donc plutôt encourageante et appelle des recherches complémentaires.

5 Conclusion

Nous avons montré que le groupe d'automorphismes d'un code correcteur peut être utilisé pour réaliser de l'augmentation de données et introduire de la diversité de représentation dans l'entraînement d'un modèle de décodeur neuronal. Nous avons aussi montré comment utiliser cette même stratégie d'augmentation pour introduire de la diversité de prédiction à l'inférence, ce qui s'apparente au décodage par permutations. On peut imaginer d'autres façons d'augmenter les données à l'entraînement, comme par exemple créer de nouvelles observations par un ré-échantillonnage judicieux du bruit sur les motifs d'erreurs MLD fournis par le jeu de données.

6 Remerciements

Ces travaux ont été financés par le projet ANR-21-CE25-0006 AI4CODE. Ils ont par ailleurs bénéficié d'un accès aux moyens de calcul de l'IDRIS au travers de l'allocation de ressources 2025-AD011016057 attribuée par GENCI.

Références

[1] A. BENNATAN, Y. CHOUKROUN et P. KISILEV : Deep learning for decoding of linear codes : A syndrome-based approach. *In IEEE ISIT 2018*, Vail, CO, USA, June 2018.

[2] Y. CHOUKROUN et L. WOLF : Error correction code transformer. *In NeurIPS 2022*, New Orleans, LO, USA, 2022.

[3] G. DE BONI ROVELLA et M. BENAMMAR : Improved syndrome-based neural decoder for linear block codes. *In IEEE GLOBECOM 2023*, Kuala Lumpur, Malaysia, Dec. 2023.

[4] M. P. C. FOSSORIER et S. LIN : Soft-decision decoding of linear block codes based on ordered statistics. *IEEE Trans. Inform. Theory*, 41(5), Sep. 1995.

[5] M. GEISELHART *et al.* : Automorphism ensemble decoding of quasi-cyclic LDPC codes by breaking graph symmetries. *IEEE Commun. Lett.*, 8(26), Aug. 2022.

[6] M. HELMLING *et al.* : Database of channel codes and ML simulation results. [Online], 2019.

[7] A. ISMAIL, R. LE BIDAN, E. DUPRAZ et C. ABDEL NOUR : Doing more with less : Towards more data-efficient syndrome-based neural decoders. *In IEEE ICMLCN 2025*, Barcelona, Spain, May 2025.

[8] F. J. MACWILLIAMS et N. J. A. SLOANE : *The Theory of Error-Correcting Codes*. North-Holland Pub, 1977.

[9] Y. SHEN *et al.* : Toward universal belief propagation decoding for short binary block codes. *IEEE J. Selec. Areas Commun.*, 43(5), Apr. 2025.

[10] J. SNYDERS et Y. BE'ERY : Maximum likelihood soft decoding of binary block codes and decoders for the golay codes. *IEEE Trans. Inform. Theory*, 35(5), Sep. 1989.