



---

# Checkpointing Efficace pour les Réseaux de Neurones Profonds

---

Bryan Chen, bryan.chen@etu.toulouse-inp.fr<sup>a</sup>,  
t0934135@u.nus.edu<sup>b</sup>

<sup>a</sup>*Institut National Polytechnique, ENSEEIHT, Toulouse, France,*

<sup>b</sup>*National University of Singapore, Singapore*

DIPLÔME D'INGÉNIER EN SCIENCE DU NUMÉRIQUE  
HIGH-PERFORMANCE COMPUTING, BIG DATA, AND ARTIFICIAL INTELLIGENCE

14/03/2024 - 14/09/2024

*Tuteur université*  
OLIVIER COTS  
olivier.cots@enseeiht.fr

*Tuteur entreprise*  
WEI TSANG OOI  
ooiwt@comp.nus.edu.sg



## REMERCIEMENTS

« SEULS, NOUS POUVONS FAIRE SI PEU ; ENSEMBLE, NOUS POUVONS FAIRE TELLEMENT »  
- HELEN KELLER



Le stage de recherche a été transformateur grâce aux conseils constants, au soutien inestimable et aux connaissances approfondies des mentors M. Ooi Wei Tsang (NUS), M. Yannis Montreuil (Sorbonne, UPMC), M. Axel Carlier (INP-ENSEEIHT) et M. Ng Lai Xing (A\*STAR). Leur expertise et leur disponibilité ont non seulement enrichi mon expérience académique, mais ont également apporté des perspectives nouvelles et innovantes à mon travail de recherche. Je suis particulièrement reconnaissant pour les discussions stimulantes et les orientations précieuses qu'ils m'ont fournies tout au long de cette période.

Je tiens également à exprimer ma gratitude envers l'ensemble de l'équipe de CNRS@Create pour leurs efforts de collaboration et leur camaraderie. Le dynamisme et l'esprit d'équipe qui règnent au sein de cette organisation ont grandement contribué à la réussite de mon stage. Les échanges avec mes collègues, toujours prêts à partager leurs idées et leurs expériences, ont été d'une aide précieuse pour surmonter les défis rencontrés et m'ont permis de progresser dans mes projets.

Un merci tout particulier à mes amis et collègues de CNRS@Create, dont la solidarité et le soutien ont rendu cette expérience non seulement enrichissante d'un point de vue professionnel, mais également agréable et mémorable sur le plan personnel. Les moments de convivialité et les discussions informelles ont été tout aussi importants que les sessions de travail, et ont forgé des liens qui, je l'espère, perureront bien au-delà de la fin de ce stage.

Enfin, je souhaite remercier toutes les personnes qui, de près ou de loin, ont contribué à la réussite de cette expérience. Leur encouragement et leur appui ont été des éléments clés de ma motivation et de mon engagement tout au long de ce parcours.

## Résumé

Le réentraînement de toutes les données accumulées depuis que l’empoisonnement des données s’est produit afin d’obtenir un nouveau modèle à jour à déployer, en particulier dans le contexte des systèmes critiques, est souvent difficile. Ainsi, une branche et un entraînement du modèle sur les données potentiellement corrompues et les données définitivement non corrompues peuvent être créés dès le départ. Cela signifie qu’un temps de récupération presque immédiat pourrait être obtenu, car une branche non corrompue peut être directement utilisée en cas d’empoisonnement des données. De plus, une branche de modèles entraînés sur les données potentiellement corrompues pourrait permettre des investigations et des études supplémentaires sur la source et les impacts des données corrompues. Bien qu’un faible temps de récupération soit obtenu, un grand nombre de modèles doivent être entraînés et stockés. Par conséquent, un système de points de contrôle et de compression approprié est nécessaire. L’objectif de cette étude est de créer un mécanisme de point de contrôle pour le processus d’entraînement des modèles dans le contexte du « branchement » – axé sur l’obtention de la meilleure performance de compression avec une précision de restauration acceptable.

**Mots clés :** Désapprentissage Machine ; Optimisation ; Compression ; Apprentissage Incrémental ; Adaptation de rank faible ; Points de contrôle

---

# Table des matières

<b>1 Contexte</b>	<b>7</b>
1.1 Présentation de CNRS@Create . . . . .	7
1.2 Présentation du programme Descartes . . . . .	7
1.3 Description du work package 4 . . . . .	9
<b>2 Organisation du travail</b>	<b>10</b>
<b>3 Méthodes de travail</b>	<b>11</b>
3.1 Analyse préliminaire et compréhension du contexte . . . . .	11
3.2 Planification et fixation d'objectifs . . . . .	11
3.3 Développement et implémentation . . . . .	12
3.4 Tests et validation . . . . .	12
3.5 Documentation et communication . . . . .	12
<b>4 Outils de travail</b>	<b>12</b>
4.1 Environnement de développement intégré (IDE) . . . . .	13
4.2 Outils de contrôle de version . . . . .	13
4.3 Bibliothèques et frameworks . . . . .	13
4.4 Environnement de virtualisation et de gestion de packages . . . . .	13
4.5 Outils de communication et de collaboration . . . . .	13
4.6 Superordinateurs et ressources matérielles . . . . .	14
4.7 Visualisation et suivi des entraînements . . . . .	14
4.8 Outils d'analyse et de visualisation des données . . . . .	14
<b>5 Introduction</b>	<b>15</b>
5.1 Sujet du stage . . . . .	15
5.2 Description des techniques utilisées . . . . .	16
5.2.1 LC-checkpoint . . . . .	16
5.2.1.1 Introduction . . . . .	16
5.2.1.2 Motivation . . . . .	16
5.2.1.3 Objectif . . . . .	16
5.2.1.4 Méthodologie . . . . .	17
5.2.1.5 Formulation . . . . .	18
5.2.1.6 Utilité de LC-Checkpoint . . . . .	19
5.2.1.7 Résumé . . . . .	19
5.2.2 LoRA : Low-Rank Adaptation - Adaptation de rang faible . . . . .	19
5.2.2.1 Introduction . . . . .	19
5.2.2.2 Motivation . . . . .	20
5.2.2.3 Architecture de LoRA . . . . .	20
5.2.2.4 Fonctionnement de LoRA . . . . .	21

5.2.2.5	Utilité de LoRA . . . . .	21
5.2.2.6	Choix du rang faible . . . . .	21
5.2.2.7	Résumé . . . . .	21
5.2.3	Delta-LoRA : Delta Low-Rank Adaptation . . . . .	22
5.2.3.1	Introduction . . . . .	22
5.2.3.2	Motivation . . . . .	22
5.2.3.3	Architecture de Delta-LoRA . . . . .	22
5.2.3.4	Fonctionnement de Delta-LoRA . . . . .	23
5.2.3.5	Utilisation de Delta-LoRA . . . . .	24
5.2.3.6	Résumé . . . . .	24
5.3	Schéma de checkpointing efficace proposé . . . . .	24
<b>6</b>	<b>Détails des résultats obtenus</b>	<b>26</b>
6.1	Reproduction des résultats . . . . .	28
6.2	Extension des résultats sur CIFAR-10 . . . . .	28
6.3	Extension des résultats avec d'autres modèles : VGG-16 Full, ResNet-50 et Vision Transformer . . . . .	29
6.3.1	Essai avec VGG-16 Full, ResNet-50 et ViT-Tiny . . . . .	29
6.3.2	Vision Transformer-Small puis étude des couches d'application du Delta-LoRA . . . . .	31
<b>7</b>	<b>Empoisonnement des données</b>	<b>33</b>
7.1	Motivation derrière l'application de l'apprentissage incrémental . . . . .	33
7.2	Hypothèses sur le découpage de l'ensemble de données . . . . .	33
7.3	Résultats avec l'apprentissage incrémental . . . . .	34
7.3.1	Étude des configurations du découpage de l'ensemble d'entraînement d'intérêt . . . . .	34
7.3.2	Étude du rang en apprentissage incrémental . . . . .	37
<b>8</b>	<b>Vérification dans les ensembles de données</b>	<b>41</b>
8.1	Vérification des images uniques . . . . .	41
8.2	Vérification de la présence de toutes les classes dans les dataloaders . . . . .	42
8.3	Synthèse . . . . .	45
<b>9</b>	<b>Explication des choix des modèles</b>	<b>45</b>
<b>10</b>	<b>Pseudo-code</b>	<b>45</b>
10.1	Checkpointing Efficace : Sans apprentissage incrémental . . . . .	46
10.2	Checkpointing Efficace : Avec apprentissage incrémental . . . . .	46
<b>11</b>	<b>Conclusion</b>	<b>47</b>
11.1	Synthèse des résultats et perspectives . . . . .	47
11.2	Compétences techniques développées . . . . .	48

## Table des figures

1.1	DesCartes Augmented Marina Bay Twin . . . . .	8
5.1	Vue d'ensemble de LC-Checkpoint . . . . .	18
5.2	Reparamétrisation avec entraînement que de A et B . . . . .	20
5.3	Différence entre Delta-LoRA et LoRA . . . . .	23
5.4	Schéma de points de reprise (checkpointing) proposé . . . . .	25
5.5	Schéma de restoration proposée . . . . .	26
5.6	Schéma de restoration proposée considérant les superstep . . . . .	26
7.1	Evolution de la performance pour LeNet-5 pendant l'apprentissage incrémental pour les différentes configurations . . . . .	36
7.2	Evolution de la performance pour ViT-B/16 pendant l'apprentissage incrémental . . . . .	36
7.3	Validation accuracy de ViT-L pré-entraîné sur ImageNet-21k, adapté à Stanford Cars . . . . .	40
7.4	Train Loss de ViT-L pré-entraîné sur ImageNet-21k, adapté à Stanford Cars . . . . .	40
8.1	Vérification de la différence des images entre l'ensemble d'entraînement et l'ensemble de test pour MNIST . . . . .	41
8.2	Vérification de la présence de toutes les classes dans chaque dataloader pour MNIST . . . . .	42
8.3	Vérification de la présence de toutes les classes dans le dataloader 1 pour Stanford Cars . . . . .	43
8.4	Vérification de la présence de toutes les classes dans le dataloader 2 pour Stanford Cars . . . . .	43
8.5	Vérification de la présence de toutes les classes dans le dataloader 3 pour Stanford Cars . . . . .	44
8.6	Vérification de la présence de toutes les classes dans le dataloader 4 pour Stanford Cars . . . . .	44
9.1	Chronologie des modèles de classification d'images . . . . .	45

## Liste des tableaux

1	Résultats de compression . . . . .	27
2	Paramètres de test . . . . .	27
3	Résultats de compression pour la réimplémentation . . . . .	28
4	Paramètres de test pour la réimplémentation . . . . .	28
5	Résultats de compression sur CIFAR-10 <sup>†</sup> . . . . .	29
6	Paramètres de test sur CIFAR-10 . . . . .	29
7	Résultats de compression sur MNIST avec VGG-16 Full, ResNet-50 et ViT-T <sup>†</sup> . . . . .	30

8	Comparaison du taux de compression sur MNIST entre ViT-S (MSAxMLP), ViT-S (MSA) et ViT-S (MLP) . . . . .	32
9	Comparaison du taux de compression sur MNIST pour LeNet-5 sur les trois configurations 4*25%, 20*5% et 80*1.25% . . . . .	35
10	Taux de compression sur CIFAR-10 pour ViT-B/16 sur 4*25% . . . . .	35
11	Comparaison des configurations de LeNet-5 et ViT-B/16 sur les différents datasets . . . . .	36
12	Comparaison des taux de compression pour différents rangs dans la configuration 4*25% avec rang = 16 et rang = 4 . . . . .	37
13	Configuration pour la comparaison des taux de compressions pour rang = 16 et 4 dans la configuration 4*25% pour ViT-B/16 sur CIFAR-10 . . . . .	37
14	Comparaison des taux de compression pour différents rangs dans la configuration 4*25% pour ViT-B/16 pré-entraîné sur ImageNet-1k, adapté sur CIFAR-10 . . . . .	38
15	Taux de compression et performance dans la configuration 4*25% pour ViT-L/16 pré-entraîné sur ImageNet-21k, adapté sur Stanford Cars . . . . .	39
16	Configuration pour le taux de compression pour rang = 8 dans la configuration 4*25% de ViT-L/16 sur Stanford Cars . . . . .	39

# 1 Contexte

## 1.1 Présentation de CNRS@Create

CNRS@Create [1] est la première filiale du CNRS, basée à Singapour, fondée en 2019, au sein de CREATE (Campus for Research Excellence and Technological Enterprise), qui a été établi par la National Research Foundation (NRF) [2] en 2006 pour accroître la vitalité et la diversité de l'écosystème de recherche et développement (R&D) de Singapour. CREATE réunit des chercheurs de dix institutions en collaboration avec la National University of Singapore (NUS), la Nanyang Technological University (NTU) et A\*STAR (Agency for Science, Technology and Research) sur quatorze programmes de recherche interdisciplinaires, dont :

- Massachusetts Institute of Technology (MIT)
- University of California Berkeley (UCB)
- Cambridge University
- ETH Zurich
- Technical University of Munich (TUM)
- Hebrew University of Jerusalem
- Centre National de la Recherche Scientifique (CNRS)
- University of Illinois at Urbana-Champaign (UIUC)
- Imperial College London (ICL)
- Shanghai Jiao Tong University (SJTU)

CNRS@Create représente un investissement total de 50 millions de dollars singapouriens, dont 25 fournis par le gouvernement singapourien et le reste par le CNRS. Plusieurs programmes y sont développé, comme Space (2 ans), Calypso (2 ans), ScanCells (2 ans), EcoCTs (2 ans) et Descartes (5 ans). CNRS@Create a pour objectifs de :

- Renforcer la position mondiale de la France et de Singapour dans les domaines de recherche ayant le plus grand potentiel pour la société présente et future.
- Créer des initiatives uniques en recherche transdisciplinaire d'excellence et en développement technologique qui n'existeraient pas dans des circonstances normales ou au sein d'un seul pays ou d'une seule institution.
- Soutenir la recherche translationnelle et l'innovation pour promouvoir l'exploitation des résultats de la recherche. La participation directe de l'industrie dans les projets de recherche est encouragée pour assurer une transformation efficace des résultats de la recherche en produits innovants.

## 1.2 Présentation du programme Descartes

Le programme Descartes comporte dix "work packages" (WP), répartis en trois piliers, avec une équipe globale de 80 chercheurs permanents, 29 doctorants et plu-

sieurs ingénieurs provenant soit des membres académiques (CNRS, ENSAM, ENS Paris-Saclay, Sorbonne Université, Grenoble INP, Université Côte d'Azur, Université Toulouse III, Université Lyon III, UGA, ENAC, Université PSL, ENSEEIHT, Université de Bordeaux, Université de Strasbourg, Université Laval, Universidad Zaragoza, Eindhoven University of Technology – TU/e) ou des membres académiques de Singapour (NUS, NTU, A\*STAR, SUSS).

Le programme DesCartes vise à développer une intelligence artificielle hybride disruptive pour servir la ville intelligente (comme un jumeau numérique de Singapour cf. FIGURE 7.2) et permettre une prise de décision optimisée dans des situations complexes rencontrées par les systèmes urbains critiques, couvrant ainsi plusieurs domaines, des sciences sociales, de la réalité virtuelle à l'utilisation intelligente de l'IA.



FIGURE 1.1 – DesCartes Augmented Marina Bay Twin

Le programme est structuré en trois piliers :

- 1. Intelligent Computing** : Développer des outils informatiques pour gérer les données intelligentes et produire des jumeaux et des contrôleurs hybrides

certifiés et explicables, basés sur des modèles physiques et des connaissances.

2. **Empowering People in Smart Societies** : Étudie le processus collaboratif de prise de décision entre les opérateurs humains et l'IA hybride, développe des interfaces en langage naturel pour une interaction plus naturelle entre les opérateurs et l'IA hybride, élabore des lignes directrices éthiques pour les décideurs politiques et les concepteurs d'IA hybride, propose un ensemble minimal de lignes directrices pour la réglementation juridique et le développement de politiques pour la mise en œuvre de l'IA hybride, et étudie les processus d'intégration et de co-développement globaux de Descartes.
3. **Engineering and Builders** : Apporte les dimensions ingénierie, technologique et applicative au concept d'IA hybride. Il vise à proposer une méthodologie opérationnelle et une architecture, de la détection intelligente et du diagnostic prédictif à un contrôle robuste et évolutif, pour des systèmes complexes et critiques.

Le programme se structure autour de cinq cas d'utilisation principaux, avec la collaboration de plusieurs industries (ESI Group, CETIM-MATCOR, EDF, ARIA Technologies, SKF, Thales, Immersion, Naval Group et Expleo) et partenaires (DSO National Laboratories, Azur Drones, Singapore Institute of Technologies - SIT) :

1. Étude des cartes de vent
2. Planification des trajectoires de drones
3. Collecte de données à distance
4. Recherche d'alternatives en cas d'urgence
5. Optimisation et prévision du réseau énergétique

### 1.3 Description du work package 4

Le WP4 s'intègre dans le pilier "Empowering People in Smart Societies" du programme Descartes et est constitué d'environ quinze scientifiques, incluant cinq chercheurs de Singapour et quatre français. Il se concentre sur la manière dont les humains peuvent interagir avec l'IA pour :

1. Apporter des aspects humains qui ne peuvent être modélisés de manière computationnelle dans les systèmes et algorithmes d'IA, formant une IA hybride avec l'interaction humaine en son cœur
2. Permettre à l'IA hybride d'augmenter la perception et la cognition humaines (en aidant notamment les humains dans la prise de décision)

Au sein de ce WP, ils développent des techniques d'interaction et de visualisation pour la collaboration entre les humains et l'IA afin de construire une IA intelligente, explicite et responsable, explorant également comment améliorer la robustesse de l'IA face aux entrées humaines erronées et malveillantes. Enfin, ils examineront

comment l'IA hybride peut augmenter la perception et la cognition humaines. Le WP4 comporte ainsi deux tâches principales :

1. **Apprendre à différer (Learning to defer)** : Impliquant des modèles de machine learning qui prennent des décisions autonomes en fonction de leur confiance et sollicitent des experts lorsque cela est nécessaire pour améliorer le modèle.
2. **Désapprendre des experts (Unlearning from experts)** : En cas de données malicieuses ou biaisées, la détection de ces dernières sont considérées comme acquise, et la question est de se demander comment enlever ces données de notre modèle de manière efficace.

## 2 Organisation du travail

Le stage s'inscrit dans le cadre du projet Descartes du CNRS@Create, sous la supervision de M. Ooi Wei Tsang (NUS), M. Axel Carlier (INP-ENSEEIHT), M. Lai Xing Ng (A\*STAR), et M. Yannis Montreuil (Sorbonne Université, UPMC).

D'un côté, le Ph.D Yannis Montreuil travaillait sur le sujet "Apprendre à différer", encadré par le Lead Principal Investigator (PI) M. Ooi Wei Tsang, les PIs M. Axel Carlier et M. Lai Xing Ng, dont le sujet de son travail est de proposer des méthodes intégrant les connaissances d'experts dans les algorithmes d'IA pour améliorer la prise de décision. Dans ce système collaboratif, les prédictions sont différées à un expert qualifié lorsque le modèle rencontre des incertitudes ou des erreurs. Ce travail se situe à l'intersection des statistiques, de l'apprentissage automatique et des sciences sociales.

De l'autre côté, le sujet de mon stage vient compléter le travail de M. Yannis Montreuil lorsque l'expert fournit des données malicieuses ou biaisées, il faut ainsi élaborer d'un framework efficace d'entraînement de modèle de machine learning, qui considère ce problème, dont la solution proposée est d'intégrer un système de checkpoints compressés, afin d'oublier la donnée malicieuse, considérant que la détection est admise.

Le travail de ce stage a été structuré de manière à équilibrer la recherche théorique et les applications pratiques. Chaque semaine, des objectifs spécifiques ont été définis pour s'assurer d'une progression continue et cohérente. Des appels hebdomadaires ont été organisés pour discuter des avancées, des défis rencontrés et des solutions envisagées.

Au début du stage, étant donné que mon superviseur avait déjà confié la tâche à un étudiant de la NUS d'implémenter ce système, j'ai commencé par reprendre le

travail de cet étudiant. Cela impliquait d'assimiler et de comprendre son code avant de développer et d'améliorer son travail. Cette phase initiale a été cruciale pour établir une base solide sur laquelle j'ai pu construire et proposer des améliorations. De plus, il s'agissait également de lire et de comprendre les articles de recherche associés aux techniques impliquées, notamment LC-checkpoint et Delta-LoRA, qui est une version modifiée de LoRA, ainsi comprendre LoRA était également essentiel pour appréhender Delta-LoRA et le schéma utilisé dans le projet.

Chaque semaine, des réunions étaient tenues auprès de M. Ooi Wei Tsang, M. Yannis Montreuil, M. Axel Carliet et M. Lai Xing Ng, pour partager les progrès réalisés, obtenir des retours et ajuster les objectifs en fonction des défis rencontrés. Ce cadre structuré a permis de maintenir une progression continue et d'assurer que les objectifs du projet étaient atteints de manière efficace et organisée.

### 3 Méthodes de travail

La méthode de travail adoptée durant ce stage s'est appuyée sur une approche systématique et itérative pour assurer une progression méthodique et des résultats de qualité. Cette approche a été structurée autour des éléments suivants :

#### 3.1 Analyse préliminaire et compréhension du contexte

La première étape a consisté à effectuer une analyse approfondie des besoins du projet et à comprendre le contexte scientifique et technique. Cela a impliqué :

- La lecture et la relecture des articles de recherche pertinents, notamment ceux portant sur LC-checkpoint, Delta-LoRA et LoRA.
- La compréhension des concepts théoriques sous-jacents et des schémas utilisés dans ces techniques.
- L'analyse des travaux antérieurs, en particulier le code et les contributions de l'étudiant de la NUS, pour en saisir les fondements et les implémentations spécifiques.

#### 3.2 Planification et fixation d'objectifs

Sur la base de l'analyse préliminaire, un plan de travail détaillé a été élaboré. Ce plan comprenait :

- La définition d'objectifs hebdomadaires spécifiques et réalisables pour maintenir une progression continue.
- La création d'un calendrier de réunions hebdomadaires pour discuter des avancées, des défis et des ajustements nécessaires.
- L'établissement de jalons intermédiaires pour suivre les progrès et ajuster les priorités en conséquence.

### 3.3 Développement et implémentation

L'étape suivante a impliqué le développement et l'implémentation des solutions techniques. Cela comprenait :

- L'assimilation et l'amélioration du code existant fourni par l'étudiant de la NUS.
- L'implémentation de nouvelles fonctionnalités et l'amélioration des algorithmes existants pour répondre aux besoins spécifiques du projet.
- La mise en place d'un système de checkpoints compressés pour traiter les données malicieuses ou biaisées fournies par les experts.

### 3.4 Tests et validation

Pour garantir la qualité et l'efficacité des solutions développées, une phase de tests et de validation rigoureuse a été menée. Cette phase incluait :

- La conception de tests unitaires pour vérifier la fonctionnalité et la fiabilité du code.
- L'analyse des résultats des tests pour identifier et corriger les éventuelles erreurs ou inefficacités.

### 3.5 Documentation et communication

Enfin, une documentation complète et claire a été élaborée pour assurer la traçabilité et la reproductibilité des travaux. Cette documentation comprenait :

- La rédaction de rapports détaillés sur les méthodologies utilisées, les résultats obtenus et les conclusions tirées.
- La création de guides d'utilisation et de manuels techniques pour faciliter la compréhension et l'utilisation des outils développés.
- La communication régulière des avancées et des résultats aux superviseurs et aux membres de l'équipe via des réunions et des présentations.

Cette méthode de travail structurée a permis de mener à bien le projet de manière efficace, en garantissant une progression continue et des résultats de qualité, tout en favorisant une collaboration active et productive au sein de l'équipe.

## 4 Outils de travail

Pour mener à bien les tâches et les objectifs du stage, une sélection appropriée d'outils et de technologies a été utilisée. Ces outils ont joué un rôle essentiel dans la gestion, le développement et l'évaluation des solutions proposées. Voici les principaux outils utilisés :

## 4.1 Environnement de développement intégré (IDE)

L'IDE choisi pour le développement du code était crucial pour assurer une productivité et une efficacité maximales. J'ai utilisé principalement :

- **Visual Studio Code** : Utilisé pour des tâches spécifiques nécessitant une flexibilité et une légèreté accrues, ainsi que pour l'édition de fichiers Markdown et la gestion de documentation.

## 4.2 Outils de contrôle de version

Pour assurer la gestion efficace du code source et faciliter la collaboration avec l'équipe de recherche, des outils de contrôle de version ont été utilisés :

- **Git** : Utilisé avec des dépôts GitHub privés pour versionner le code, suivre les modifications, gérer les branches de développement et faciliter la collaboration avec d'autres membres de l'équipe.
- **GitHub** : Plateforme utilisée pour héberger les dépôts Git, gérer les issues, et faciliter les discussions et les revues de code au sein de l'équipe.

## 4.3 Bibliothèques et frameworks

Pour le développement des algorithmes et la manipulation des données, plusieurs bibliothèques et frameworks ont été essentiels :

- **NumPy** : Pour la manipulation efficace des données et le calcul numérique en Python.
- **PyTorch** : Pour l'implémentation et l'entraînement des modèles d'apprentissage profond.
- **Hugging Face Transformers** : Pour travailler avec des modèles pré-entraînés et développer des algorithmes avancés en utilisant les transformers.

## 4.4 Environnement de virtualisation et de gestion de packages

Pour maintenir un environnement de développement propre et gérer les dépendances logicielles, les outils suivants ont été utilisés :

- **Conda** : Utilisé pour la gestion des environnements de données et pour installer des packages spécifiques nécessaires au projet.

## 4.5 Outils de communication et de collaboration

Pour faciliter la communication et la collaboration avec les superviseurs et les membres de l'équipe, plusieurs outils ont été utilisés :

- **Zoom** : Pour les réunions virtuelles hebdomadaires, les discussions et les mises à jour sur l'avancement du projet.

- **Slack** : Utilisé pour les communications instantanées, la gestion des fichiers et le partage rapide d'informations au sein de l'équipe.
- **Google Slides** : Pour créer des présentations détaillées des résultats lors des réunions et des meetings, permettant une visualisation claire et interactive des progrès et des découvertes du projet.

## 4.6 Superordinateurs et ressources matérielles

Pour les calculs intensifs nécessaires à l'entraînement des modèles de machine learning, plusieurs ressources matérielles ont été mises à disposition :

- **Superordinateurs ASPIRE 2A du NSCC** : Composés de 4 NVIDIA A100 40G SXM et des AMD EPYC 7713, avec communication via PBS pour l'interaction avec le serveur.
- **VPN à NUS** : Utilisé pour se connecter de manière sécurisée au serveur de calcul.
- **Ordinateur du CNRS@Create** : Équipé de deux NVIDIA GeForce RTX 3090 pour des calculs supplémentaires.
- **Ordinateur personnel** : Équipé d'un Intel(R) UHD Graphics i7 de 12e génération et d'une NVIDIA GeForce RTX 4060 Laptop GPU pour le développement local et les tests.
- **PuTTY** : Utilisé pour se connecter au serveur de NSCC.
- **Commandes PBS (Portable Batch System)** : Utilisés pour lancer des tâches .sh sur le serveur de NSCC.
- **FileZilla** : Utilisé pour transférer des fichiers de mon ordinateur local au serveur de NSCC.

## 4.7 Visualisation et suivi des entraînements

Pour la visualisation des mesures de l'entraînement des réseaux de neurones :

- **TensorBoard** : Initialement utilisé pour la visualisation des métriques d'entraînement.
- **Weights Biases (wandb)** : Choisi par la suite pour sa simplicité d'utilisation et son interface utilisateur intuitive, facilitant le suivi et la comparaison des expériences, et préféré à TensorBoard pour une meilleure expérience utilisateur.

## 4.8 Outils d'analyse et de visualisation des données

Pour comparer plusieurs configurations de modèles de machine learning, notamment les Vision Transformers (ViT), les outils suivants ont été utilisés :

- **R (librairie ggplot2)** : Utilisé pour créer des graphiques et comparer différentes configurations de modèles de machine learning, permettant une analyse

visuelle claire et détaillée des performances.

Ces outils ont joué un rôle crucial dans la gestion efficace du projet, en facilitant la collaboration, en assurant la traçabilité des développements et en permettant une gestion agile des processus de développement et d'évaluation.

## 5 Introduction

### 5.1 Sujet du stage

Le stage se concentre sur le WP4, traitant de la collaboration robuste Humain-IA, dont l'objectif est d'intégrer des experts humains dans la boucle d'apprentissage lorsque le modèle IA manque de précision. En cas de biais humain détecté ou de données malicieuses, on cherche à faire que l'IA "désapprenne" de cette dernière, une approche nommée "unlearning from experts".

L'approche naïve serait de tout apprendre, une nouvelle fois, à partir de zéro, mais cela serait coûteux en terme d'espace de stockage, en coût de calcul et ainsi en temps.

Une approche moins coûteuse, qui est celle sur laquelle le stage se concentre, porte sur des techniques de checkpoints compressés sauvegardés pendant l'entraînement du modèle. Lorsqu'il y a une détection d'une donnée malicieuse pendant l'entraînement, l'idée serait de revenir sur la branche (point de contrôle) antérieure à l'occurrence de la détection, puis de continuer l'entraînement à partir de cette branche. Pour compresser les points de contrôle, l'idée du stage est de combiner deux algorithmes complémentaires, compressant ces points de contrôle, afin de rendre la sauvegarde de ces derniers moins coûteuse en stockage et en temps, contrairement à l'idée qui serait d'enregistrer tous les poids du modèle pour chaque branche.

Concrètement, il s'agit d'appliquer un premier algorithme LC-checkpoint [3], composé d'une quantification à base d'exponentielle (exponent-based quantization) puis d'une promotion prioritaire (priority promotion) et enfin d'une compression sans perte d'information via un encodage de Huffman, combiné avec le deuxième algorithme Delta Low-Rank Adaptation (Delta-LoRA) [4], afin de réduire davantage la taille des checkpoints. Ces algorithmes seront décrits plus en détail dans la sous-section suivante.

L'idée du stage est ainsi de vérifier la pertinence de l'enchaînement des deux techniques, dans laquelle une première idée serait de faire un Proof of Concept (POC) sur des modèles de classification d'images simples : LeNet [5], AlexNet [6], VGG16 [7] Lite Version, puis ensuite d'étendre cela à des modèles plus coûteux, plus ré-

cents, comme VGG16 [7] Full Version, ResNet-50 [8], Vision Transformer (ViT) [9]. Il s'agit également de faire ces vérifications sur des bases de données de plus en plus grandes et complexes, avec d'abord MNIST [10], puis ensuite d'élargir à CIFAR-10 [11], puis éventuellement CIFAR-100 [12] et ImageNet [13].

Ensuite, si les résultats s'avèrent pertinents et si le temps le permet, l'idée serait également de développer une API accessibles aux chercheurs de CNRS@Create afin que les autres WP du programme Descartes puissent utiliser cette technologie pour entraîner leur modèle de machine learning.

## 5.2 Description des techniques utilisées

Dans le cadre de ce stage de recherche, une approche innovante est proposée pour optimiser la compression des checkpoints en combinant deux techniques avancées : LC-checkpoint (Lossy Compression Checkpoint) et Delta-LoRA. Avant de détailler cette méthode hybride, il est essentiel de comprendre individuellement le fonctionnement de chaque algorithme.

### 5.2.1 LC-checkpoint

#### 5.2.1.1 Introduction

L'architecture de LC-checkpoint [14] (Lossy Compression Checkpoint) telle qu'introduite dans l'article de recherche "On Efficient Constructions of Checkpoints" publié en 2020 par Yu Chen, Zhenming Liu, Bin Ren, et Xin Jin a pour objectif principal de proposer une solution efficace pour optimiser la gestion des points de reprise (checkpoints) dans les systèmes informatiques en cas d'une panne (comme l'explosion du gradient [15] et la division par zéro [16]) et pour la sauvegarde des états intermédiaires. Cette section explique l'architecture de LC-checkpoint, la motivation derrière cette méthode, son utilité, et son fonctionnement.

#### 5.2.1.2 Motivation

La sauvegarde régulière des états d'un programme, connue sous le nom de checkpointing, est essentielle pour garantir la récupération en cas de panne. Cependant, les méthodes traditionnelles de checkpointing peuvent être coûteuses en termes de temps et d'espace de stockage. LC-checkpoint propose une approche de compression avec perte (lossy compression) pour réduire ces coûts, tout en maintenant une précision acceptable des données sauvegardées.

#### 5.2.1.3 Objectif

L'objectif de LC-checkpoint est de minimiser le volume de données à sauvegarder et le temps nécessaire pour revenir au dernier point d'entraînement avant l'occurrence

de la panne, tout en conservant l'intégrité et la validité des données nécessaires pour la reprise du programme.

#### 5.2.1.4 Méthodologie

LC-checkpoint (cf. FIGURE 5.1) utilise plusieurs techniques pour compresser efficacement les données, utilisant un schéma de codage différentiel ou d'encodage delta, qui ne suit que les différences entre deux points de contrôle. L'algorithme commence par quantifier avec l'exponent-based quantization sur les différences entre deux valeurs successives, puis élimine celles dont les mises à jour sont sans conséquence avec la technique de Priority promotion, et enfin compresse encore davantage à l'aide d'un encodage de Huffman. Un développement de ce que font chacune de ces techniques :

**Delta-Encoding Scheme :** Le schéma de codage différentiel ou d'encodage delta (delta-encoding scheme) est une technique qui stocke les différences (deltas) entre les valeurs successives plutôt que les valeurs elles-mêmes. Cette approche est particulièrement efficace lorsque les données présentent une certaine continuité ou corrélation entre les états successifs. Les deltas sont généralement plus petits et comportent beaucoup de valeurs nulles, pouvant ainsi être compressés plus efficacement que les valeurs brutes.

**Exponent-based Quantization :** Cette technique consiste à représenter les valeurs en utilisant une base exponentielle, ce qui permet de réduire la précision des valeurs tout en conservant une représentation approximative suffisante pour la plupart des applications. En ajustant la base exponentielle, il est possible de contrôler le niveau de compression et la perte d'information.

**Priority Promotion :** La promotion de priorité (Priority Promotion) est une technique qui donne la priorité à certaines données plus critiques lors de la compression. Les données ayant un impact plus significatif sur la précision ou la performance du programme sont compressées avec moins de perte, tandis que les données moins critiques peuvent être compressées plus agressivement.

**Huffman Encoding :** L'encodage de Huffman est une méthode de compression sans perte qui utilise des codes de longueur variable pour représenter les données. Les données les plus fréquentes sont représentées par des codes plus courts, tandis que les données moins fréquentes sont représentées par des codes plus longs. Cela permet de réduire la taille totale des données compressées.

Les données compressées en sortie de l'encodage de Huffman seront sauvegardées, permettant une reprise plus efficace de l'entraînement si ce dernier tombe en

panne pour diverses raisons. Cela nécessite la construction de "points d'arrêt", semblables à ceux utilisés pour déboguer les programmes informatiques, afin que les chercheurs puissent facilement revenir à l'état juste avant que le modèle "plante" pendant l'entraînement. En cas de panne, afin de revenir à l'état juste avant que le modèle "plante", on décomprime chaque donnée compressée sauvegardée puis on l'ajoute au modèle initial.

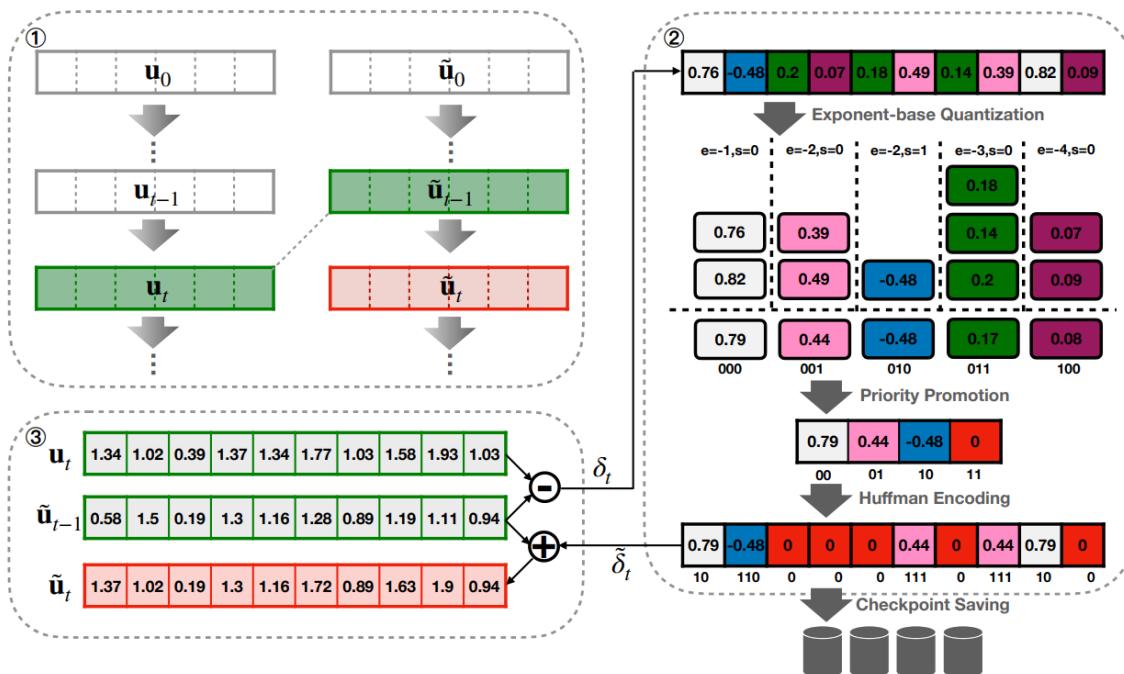


FIGURE 5.1 – Vue d'ensemble de LC-Checkpoint

### 5.2.1.5 Formulation

Plus précisément, étant donné un état du modèle  $u$  et un point optimal local  $u^*$ , l'algorithme calcule la distance de mise à jour  $\delta$ , quantifie cette distance, la compressse à l'aide du codage de Huffman et l'enregistre sur le disque, puis met à jour l'état du point de contrôle. À chaque étape, le système maintient une approximation  $\tilde{u}_t$  de l'état de vérité au sol. Nous définissons simplement, où  $u_0$  est l'état initial du modèle. Notre système maintient et met à jour en permanence  $\tilde{u}_t$ ,

$$\tilde{u}_t = u_0 + \sum_{i \leq t} \tilde{\delta}_i$$

- Pour la quantification, la compression se fait en deux étapes :
  - Quantification basée sur l'exposant (Exponent-based Quantization)**
    - Répartir les entrées de  $\delta_t$  en plusieurs groupes en fonction de l'exposant et des signes identiques.

- Représenter chaque groupe par la moyenne des valeurs maximales et minimales.
- **Promotion de la priorité (Priority Promotion)**
  - Lorsque  $\delta_{t,i} \approx 0$  ( $= u_{i,t-1} \approx u_{i,t}$ ), il est plus efficace de regrouper les mises à jour.
  - Il conserve les  $2^x - 1$  buckets avec le plus grand e (exposant) et fusionne les autres buckets en un seul avec la valeur 0.

### 5.2.1.6 Utilité de LC-Checkpoint

LC-checkpoint est particulièrement utile pour :

- **Réduction de la taille de stockage** : Diminue la quantité d'espace disque nécessaire pour stocker les checkpoints.
- **Amélioration de l'efficacité** : Réduit le temps et les ressources nécessaires pour gérer les points de reprise.
- **Tolérance aux pannes** : Maintient la capacité de reprise efficace même avec des données compressées.

### 5.2.1.7 Résumé

L'architecture de LC-checkpoint propose une solution innovante pour le problème du checkpointing. En combinant des techniques de compression avec perte et sans perte, telles que l'exponent-based quantization, la priority promotion, l'Huffman encoding, et le delta-encoding scheme, LC-checkpoint parvient à réduire significativement les coûts de stockage et de temps tout en maintenant une précision acceptable des données sauvegardées. Contrairement à la compression sans perte, qui conserve toutes les données originales, la compression avec perte élimine certaines informations redondantes ou moins importantes, permettant une réduction significative de la taille des données à stocker.

## 5.2.2 LoRA : Low-Rank Adaptation - Adaptation de rang faible

### 5.2.2.1 Introduction

Low-Rank Adaptation (LoRA) [17] est une méthode de fine-tuning, introduite par Edward J. Hu et al. en 2021 pour améliorer l'efficacité de l'adaptation (=fine-tuning) des modèles de grande taille, notamment les modèles de fondation comme les grands modèles de langage (LLMs). Cette méthode permet d'adapter les modèles en ajoutant un nombre limité de paramètres tout en conservant la performance de ces derniers. En d'autres termes, LoRA peut utiliser moins de poids que le nombre total de poids et est motivée par un article publié en 2018 qui traite de la dimensionnalité intrinsèque des grands modèles [18], affirmant qu'il existe une paramétrisation de faible dimension. L'idée de LoRA est de supposer que, lors de l'adaptation, le

changement de poids du modèle  $\delta W$  a une faible dimension intrinsèque pour mettre à jour la matrice de poids du modèle pré-entraîné figé  $W$  de taille  $d \times k$  :

$$W + \delta W = W + BA \quad \text{avec} \quad B \in R^{d \times r}, A \in R^{r \times k}, r \ll \min(d, k)$$

Cette section explique l'architecture de LoRA, la motivation derrière cette méthode, son utilité, et son fonctionnement.

### 5.2.2.2 Motivation

Avec l'augmentation de la taille des modèles de langage, le coût de leur adaptation (fine-tuning) devient prohibitif en termes de ressources computationnelles et de mémoire. LoRA a été développé pour répondre à ce problème. L'idée principale est de réduire le nombre de paramètres à ajuster lors du fine-tuning, ce qui permet d'économiser des ressources tout en maintenant, voire améliorant, les performances du modèle.

### 5.2.2.3 Architecture de LoRA

L'idée de LoRA (cf. FIGURE 5.2) est d'introduire des modifications de bas rang (low-rank) dans les poids des couches d'un modèle pré-entraîné, sans toucher aux poids originaux. Plus précisément, au lieu de mettre à jour directement les matrices de poids  $W_0$  d'une couche, LoRA les décompose en deux matrices de rang inférieur  $A$  et  $B$ , telles que :

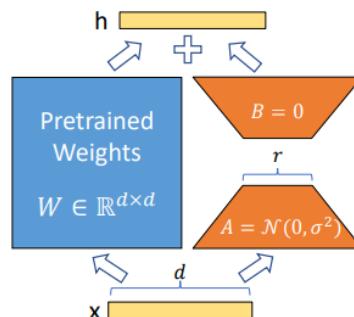


FIGURE 5.2 – Reparamétrisation avec entraînement que de A et B

$$W = W_0 + \Delta W \quad \text{avec} \quad \Delta W = A \times B$$

où  $A \in R^{d \times r}$  et  $B \in R^{r \times k}$ , avec  $r \ll \min(d, k)$ ,  $d$  étant la dimension d'entrée et  $k$  la dimension de sortie de la couche.

L'adaptation se fait uniquement sur les matrices  $A$  et  $B$ , qui contiennent beaucoup moins de paramètres que  $W_0$ . Les paramètres  $W_0$  restent figés, ce qui réduit considérablement la mémoire nécessaire et le coût computationnel du fine-tuning. En outre, LoRA peut être appliquée à n'importe quel réseau neuronal dense.

### 5.2.2.4 Fonctionnement de LoRA

Le fonctionnement de LoRA peut être résumé en plusieurs étapes :

1. **Initialisation** : Les matrices  $A$  et  $B$  sont initialisées aléatoirement ou par un schéma spécifique.
2. **Formation des poids** : Pendant la phase d'entraînement, les poids modifiés  $\Delta W = A \times B$  sont ajoutés aux poids pré-entraînés  $W_0$  pour former les poids effectifs  $W$  de la couche.
3. **Fine-tuning** : Seules les matrices  $A$  et  $B$  sont mises à jour lors du fine-tuning, tandis que  $W_0$  reste inchangé.
4. **Inference** : Lors de l'inférence, les poids effectifs  $W$  sont utilisés sans nécessiter de ressources supplémentaires significatives.

### 5.2.2.5 Utilité de LoRA

LoRA est particulièrement utile pour :

- **Réduction de la mémoire** : Moins de paramètres à stocker et à mettre à jour, ce qui réduit la mémoire nécessaire.
- **Efficacité computationnelle** : Moins de calculs nécessaires lors du fine-tuning, permettant une adaptation plus rapide et moins coûteuse en termes de ressources.
- **Flexibilité** : Possibilité de transférer et d'adapter facilement des modèles pré-entraînés à de nouvelles tâches avec un minimum de ressources.

### 5.2.2.6 Choix du rang faible

Il a été montré expérimentalement qu'un rang de 8 était un bon équilibre entre compression et performance pour des grands modèles comme LLaMa [19]. En effet, le constat est que lorsque le rang de la décomposition LoRA pour un modèle LLaMA est de 8 et plus, la performance de la LoRA ne diffère pas de manière significative. Bien que nous ne travaillions pas avec LLaMA, nous l'utilisons comme point de départ et définissons notre rang  $r$  de la décomposition LoRA d'une couche linéaire  $W^{m \times n}$  dans notre mécanisme de point de contrôle comme suit :

$$r = \min(\min(m, n), 8)$$

Des études supplémentaires pourraient être faites pour voir si d'autres valeurs de rang seraient envisageables.

### 5.2.2.7 Résumé

Low-Rank Adaptation (LoRA) est une méthode innovante qui permet d'adapter efficacement les grands modèles de langage tout en réduisant les coûts en ressources.

Cette approche est non seulement économiquement avantageuse mais également pratique pour les applications nécessitant des adaptations rapides et efficaces. La méthode, proposée par Edward J. Hu et ses collègues en 2021, adresse le problème crucial de la scalabilité dans le fine-tuning des modèles de grande taille.

### 5.2.3 Delta-LoRA : Delta Low-Rank Adaptation

#### 5.2.3.1 Introduction

Delta-LoRA [20] est une variante modifiée de LoRA (Low-Rank Adaptation), proposée par Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, et Lei Zhang dans leur article intitulé "Delta-LoRA : Fine-Tuning High-Rank Parameters with the Delta of Low-Rank Matrices". Cette méthode vise à optimiser les paramètres de haute dimension en utilisant la différence des matrices de faible rang. La mise à jour des paramètres se fait de la manière suivante :

$$W^{(t+1)} = W^{(t)} + \Delta AB \quad \text{avec} \quad \Delta AB = A^{(t+1)}B^{(t+1)} - A^{(t)}B^{(t)}$$

Delta-LoRA développe l'idée de ne pas seulement mettre à jour les matrices de faible rang A et B, mais également de propager l'adaptation aux poids pré-entraînés  $W$  via des mises à jour utilisant la différence du produit de deux matrices ( $A$  et  $B$ ) consécutives de faible rang ( $A^{(t+1)}B^{(t+1)} - A^{(t)}B^{(t)}$ ).

#### 5.2.3.2 Motivation

La motivation principale derrière Delta-LoRA est de résoudre les problèmes de surajustement et d'efficacité computationnelle dans les réseaux neuronaux de grande taille. Les méthodes traditionnelles de fine-tuning nécessitent souvent une grande quantité de mémoire et de puissance de calcul. En utilisant des matrices de faible rang et en se concentrant sur les différences entre les mises à jour, Delta-LoRA permet de réduire les besoins en ressources tout en maintenant ou en améliorant les performances du modèle.

Bien que LoRA ait permis de réduire significativement le coût du fine-tuning des modèles de grande taille, il reste des cas où les paramètres de bas rang ne suffisent pas à capturer toute la complexité des adaptations nécessaires. Delta-LoRA a été développé pour répondre à ce problème en permettant une adaptation plus fine et précise des modèles, tout en conservant les avantages de la réduction des coûts en ressources.

#### 5.2.3.3 Architecture de Delta-LoRA

L'architecture de Delta-LoRA repose sur l'idée d'incorporer les différences des produits de matrices de faible rang pour mettre à jour les paramètres de haute

dimension. Plus précisément, Delta-LoRA introduit deux matrices de faible rang  $A$  et  $B$  pour chaque couche du réseau, et met à jour les paramètres de la couche en utilisant la différence entre les produits des matrices à deux itérations consécutives :

$$\Delta AB = A^{(t+1)}B^{(t+1)} - A^{(t)}B^{(t)}$$

Cette approche permet une mise à jour plus fine et contrôlée des paramètres, évitant les sauts brusques qui peuvent entraîner une divergence lors de l'apprentissage.

L'idée principale de Delta-LoRA est d'introduire des modifications de haut rang (high-rank) en utilisant les différences (deltas) des matrices de bas rang. Au lieu de simplement ajuster les matrices de bas rang comme dans LoRA, Delta-LoRA affine également les différences entre ces matrices et leur état initial.

Concrètement, Delta-LoRA ajoute un terme de correction et met à jour  $W$  de la manière suivante :

$$W^{(t+1)} = W^{(t)} + \Delta AB \quad \text{avec} \quad \Delta AB = A^{(t+1)}B^{(t+1)} - A^{(t)}B^{(t)}$$

Contrairement à LoRA, dans Delta-LoRA, la matrice de poids pré-entraînée  $W_0$  peut également varier et être mise à jour pendant le processus de fine-tuning (cf. FIGURE 5.3).

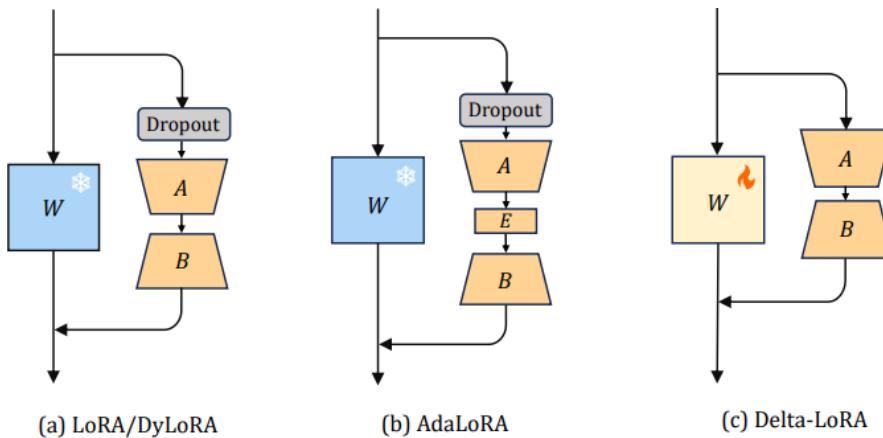


FIGURE 5.3 – Différence entre Delta-LoRA et LoRA

#### 5.2.3.4 Fonctionnement de Delta-LoRA

Le fonctionnement de Delta-LoRA peut être résumé en plusieurs étapes clés :

1. Initialisation des matrices de faible rang  $A$  et  $B$  pour chaque couche du réseau.
2. À chaque itération, calcul des produits  $A^{(t)}B^{(t)}$  et  $A^{(t+1)}B^{(t+1)}$ .
3. Calcul de la différence  $\Delta AB = A^{(t+1)}B^{(t+1)} - A^{(t)}B^{(t)}$ .
4. Mise à jour des paramètres de la couche avec  $W^{(t+1)} = W^{(t)} + \Delta AB$ .

Ce processus permet de capturer les changements subtils et pertinents dans les paramètres, améliorant ainsi la convergence et les performances du modèle.

### 5.2.3.5 Utilisation de Delta-LoRA

Delta-LoRA peut être utilisé dans divers contextes de fine-tuning de modèles de grande taille, notamment dans les domaines du traitement du langage naturel, de la vision par ordinateur, et de la reconnaissance vocale. Grâce à sa capacité à réduire les besoins en mémoire et en calcul, cette méthode est particulièrement adaptée aux environnements à ressources limitées, tout en maintenant des performances élevées.

L'utilisation de Delta-LoRA s'applique notamment pour bénéficier de :

- **L'amélioration de la précision** : Permet de capturer des variations plus fines dans les paramètres, améliorant ainsi la performance du modèle sur des tâches spécifiques.
- **L'efficacité mémoire** : Bien que plus complexe que LoRA, Delta-LoRA reste plus efficace en termes de mémoire que le fine-tuning complet des modèles.
- **La flexibilité** : Facilite l'adaptation des modèles pré-entraînés à de nouvelles tâches nécessitant des ajustements plus précis.

### 5.2.3.6 Résumé

Delta-LoRA propose une approche innovante pour la mise à jour des paramètres de modèles de grande taille en utilisant les différences des produits de matrices de faible rang. Cette méthode combine les avantages de la Low-Rank Adaptation (LoRA) avec la capacité de capturer des variations plus fines dans les paramètres du modèle grâce aux deltas des matrices de bas rang, offrant une solution efficace et performante pour le fine-tuning de réseaux neuronaux complexes. Delta-LoRA permet d'améliorer l'efficacité et la précision de l'adaptation des modèles de grande taille tout en réduisant les coûts en ressources.

## 5.3 Schéma de checkpointing efficace proposé

La proposition de ce stage consiste à intégrer les forces respectives de LC-checkpoint et Delta-LoRA dans une seule méthode de compression optimisée. Cette approche hybride vise à maximiser les avantages de chaque technique, offrant ainsi une solution plus robuste pour la gestion des checkpoints dans l'entraînement des modèles (cf. FIGURE 5.4).

La motivation de ce schéma est d'avoir des points de reprise encore plus compressés qu'en compressant uniquement avec LC-checkpoint ou Delta-LoRA. Comme LC-checkpoint, ce schéma applique au modèle pré-entraîné auquel on y ajoute des

couches de Delta-LoRA, les différentes techniques utilisés dans LC-checkpoint, soit le calcul des valeurs consécutives à l'aide du schéma de codage différentiel ou d'encodage delta (Delta-Encoding Scheme), où on applique à chaque delta la quantification à base exponentielle (Exponent-based quantization), puis on applique une promotion de priorité (Priority Promotion) pour donner plus d'importance aux informations plus significatifs, et enfin on applique un encodage de Huffman (Huffman Encoding).

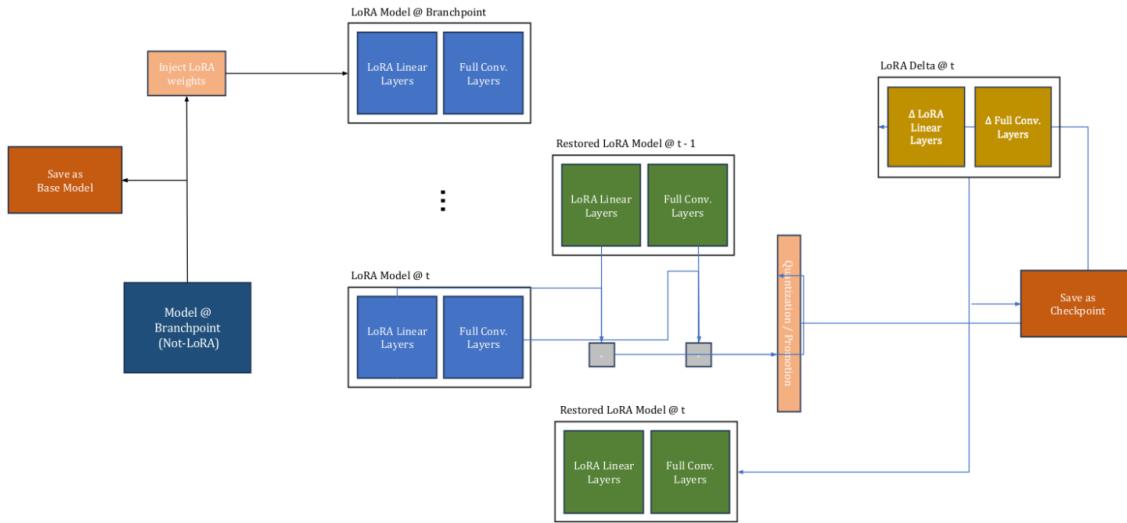


FIGURE 5.4 – Schéma de points de reprise (checkpointing) proposé

Pour être plus précis, à la place de l'encodage de Huffman, l'idée est d'utiliser GZip, qui est une méthode de compression sans perte (lossless compression), et qui en plus est directement implémentable en Python, langage de programmation utilisé pour ce sujet.

GZip est une méthode qui combine l'encodage de Huffman [21], une LZ77 [22] (une autre méthode de compression sans perte, faisant honneur à Lempel et Ziv en 1977, qui analyse les données d'entrée et détermine comment réduire la taille de ces données en remplaçant les informations redondantes par des métadonnées) et un codage par plage, autrement appelé Run-Length-Encoding ou RLE (qui est aussi un algorithme de compression de données sans perte, consistant à remplacer les suites de valeurs identiques par une paire composée du nombre de répétition et de la valeur à répéter).

Dans le schéma 5.4, on peut voir en orange la sauvegarde du modèle auquel on remplace les couches denses par des couches Delta-LoRA, mais également la sauvegarde de chaque checkpoint après compression pendant l'adaptation.

Pour la restauration en cas de panne (cf. FIGURE 5.5), on prend le modèle au dernier branchpoint puis on y ajoute la valeur après décompression des checkpoints

compressés jusqu'au dernier précédent la panne.

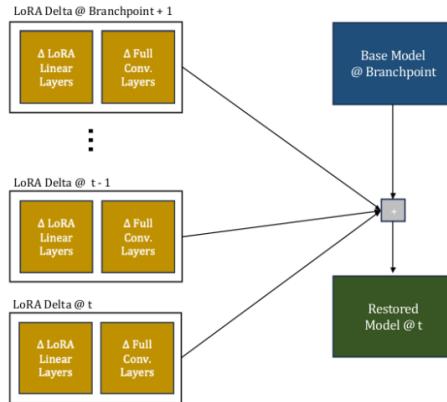


FIGURE 5.5 – Schéma de restauration proposée

Par ailleurs, pour gagner en temps de calcul pour la restauration, on mettra à jour le modèle à des itérations de manière périodique, que l'on appellera **superstep**. En effet, à tous les superstep, le modèle avec les couches de Delta-LoRA est sauvegardé (on appellera cela un **full snapshot**). De cette manière, en cas de panne, on prend le modèle au dernier superstep précédent la panne, puis s'il y a des checkpoints (des différences) après ce superstep, on les rajoute jusqu'au dernier checkpoint précédent la panne (cf. FIGURE 5.6).

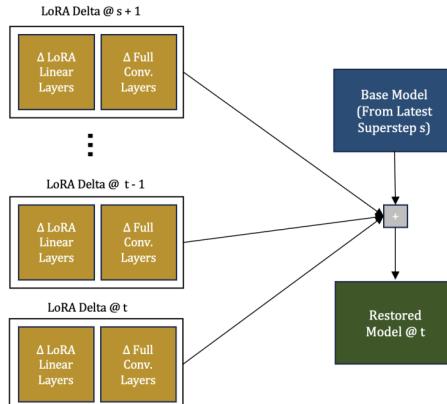


FIGURE 5.6 – Schéma de restauration proposée considérant les superstep

## 6 Détails des résultats obtenus

Des résultats préliminaires sur MNIST utilisant LeNet, AlexNet et VGG-16 Lite (une version avec un seul canal de couleur et non trois alors que VGG-16 était conçu pour être entrainé sur ImageNet qui contient des images en couleur) ont montré des performances comparables aux modèles originaux finement ajustés, ainsi

qu'une compression substantielle des points de contrôle. De la même manière, des travaux ont été réalisés sur CIFAR-10 avec des modèles avec plus de paramètres (plus grands) et plus complexes comme VGG-16 Full, Vision Transformer, ResNets.

Afin de voir l'intérêt de ce schéma de compression, l'ancien étudiant de la NUS avait montré l'intérêt de ce dernier en travaillant sur un ensemble d'entraînement, comportant les 1000 premières images de l'ensemble d'entraînement de MNIST, puis validant sur les 1000 secondes images de l'ensemble d'entraînement de MNIST, obtenant les résultats suivants :

<b>Model</b>	<b>Mechanism</b>	<b>Compression Ratio</b>	<b>Space Savings</b>	<b>Final Accuracy (Restored / Full)</b>
AlexNet	LC	808.35%	87.629%	98.4% / 98.7% (-0.3%)
	LC + dLoRA	25995.409%	99.615%	95.3% / 98.7% (-3.4%)
VGG-16	LC	813.74%	87.711%	99.1% / 99.4% (-0.3%)
	LC + dLoRA	4188.412%	97.612%	98.4% / 99.4% (-1.0%)
LeNet	LC	537.584%	81.39%	95.9% / 95.9% (-0.0%)
	LC + dLoRA	1889.2869%	94.707%	93.8% / 95.9% (-2.1%)

TABLE 1 – Résultats de compression

Les résultats sont obtenus dans les conditions suivantes :

<b>Model</b>	<b>AlexNet</b>	<b>VGG-16</b>	<b>LeNet-5</b>
Branching Point	80.72%	72.85%	77.75%
Dataset	MNIST	MNIST	MNIST
Bit-width	3	3	3
LoRA Scaling	0.5	0.5	0.5
Batch Size	32	32	32
Learning Rate	0.01	0.01	0.01
Epochs	20	20	20
Super-Step	Every 10 iteration	Every 10 iteration	Every 10 iteration
Optimizer	SGD <sup>¶</sup>	SGD <sup>¶</sup>	SGD <sup>¶</sup>

TABLE 2 – Paramètres de test

Pour avoir un modèle pré-entraîné, pour les modèles LeNet-5, AlexNet, VGG-16 Lite, on entraînait le modèle en question sur tout l'ensemble de donnée d'entraînement sans considérer les matrices A et B du Delta-LoRA jusqu'à obtenir une accuracy sur tout l'ensemble de test d'autour de 70%-80%. Ensuite, on utilise ce modèle pré-entraîné pour l'adaptation (fine-tuning) sur l'ensemble de donnée d'intérêt.

1. ¶ SGD ou Stochastic Gradient Descent, inventé par Robbins H. et Monro S. [23]

## 6.1 Reproduction des résultats

Étant donné que l'étudiant de la NUS avait commencé ce travail et avait obtenu des premiers résultats, il me fallait d'abord reproduire ses résultats, mais comme il manquait quelques fichiers essentiels à l'exécution du code lorsque cet étudiant m'a donné son code, il me fallait d'abord réimplémenter les parties manquantes. Puis, après avoir réimplémenté les parties manquantes, j'ai obtenu les résultats suivants pour les mêmes modèles utilisés, adapté (fine-tuned) sur les 1000 premières images de l'ensemble d'entraînement de MNIST et validé sur les 1000 images suivantes de l'ensemble d'entraînement de MNIST :

Model	Mechanism	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
AlexNet	LC	813.32%	87.705%	98.2% / 99.0% (-0.8%)
	LC + dLoRA	28865.393%	99.654%	96.2% / 99.0% (-3.8%)
VGG-16	LC	813.705%	87.711%	99.1% / 99.4% (-0.3%)
	LC + dLoRA	4185.214%	97.611%	98.1% / 99.4% (-1.3%)
LeNet	LC	562.256%	82.215%	96.1% / 96.1% (-0.0%)
	LC + dLoRA	1885.7610%	94.697%	91.2% / 96.1% (-4.9%)

TABLE 3 – Résultats de compression pour la réimplémentation

Les résultats sont obtenus dans les conditions suivantes :

Model	AlexNet	VGG-16	LeNet-5
Branching Point	80.5%	72.85%	76.8%
Dataset	MNIST	MNIST	MNIST
Bit-width	3	3	3
LoRA Scaling	0.5	0.5	0.5
Batch Size	32	32	32
Learning Rate	0.01	0.01	0.01
Epochs	20	20	20
Super-Step	Every 10 iteration	Every 10 iteration	Every 10 iteration
Optimizer	SGD	SGD	SGD

TABLE 4 – Paramètres de test pour la réimplémentation

## 6.2 Extension des résultats sur CIFAR-10

Après avoir obtenu les résultats de compression sur MNIST, j'ai évalué le taux de compression ainsi que l'accuracy sur CIFAR-10 avec les mêmes modèles :

Model	Mechanism	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
AlexNet	LC	634.431%	84.238%	99.6% / 99.6% (-0.0%)
	LC + dLoRA	23491.97%	99.574%	94.5% / 99.6% (-5.1%)
VGG-16	LC	814.574%	87.724%	100.0% / 100.0% (-0.0%) <sup>‡</sup>
	LC + dLoRA	4756.479%	97.898%	98.2% / 100.0% (-1.8%) <sup>‡</sup>
LeNet	LC	415.728%	81.061%	98.4% / 98.4% (-0.0%)
	LC + dLoRA	1698.463%	94.112%	86.0% / 98.4% (-12.4%)

TABLE 5 – Résultats de compression sur CIFAR-10<sup>†</sup>

Ces résultats sont obtenus dans les conditions suivantes :

Model	AlexNet	VGG-16	LeNet-5
Branching Point	80.0%	72.1%	72.0%
Dataset	CIFAR10	CIFAR10	CIFAR10
Bit-width	3	3	3
LoRA Scaling	0.5	0.5	0.5
Batch Size	32	32	32
Learning Rate	0.01	0.1	0.01
Epochs	20	20	20
Super-Step	Every 10 iteration	Every 10 iteration	Every 10 iteration
Optimizer	SGD	SGD	SGD

TABLE 6 – Paramètres de test sur CIFAR-10

### 6.3 Extension des résultats avec d'autres modèles : VGG-16 Full, ResNet-50 et Vision Transformer

L'objectif de cette sous-section est d'évaluer la portabilité de notre approche de compression sur des architectures de réseaux de neurones plus complexes et récentes, afin de déterminer si les gains en terme de compression observés avec les modèles initiaux pouvaient être reproduits dans des contextes variés. Ainsi, nous avons choisi de tester trois modèles populaires et structurellement différents : VGG-16 Full Version, ResNet-50, et le Vision Transformer (ViT).

#### 6.3.1 Essai avec VGG-16 Full, ResNet-50 et ViT-Tiny

Pour des raisons d'optimisation du temps à disposition, nous avons maintenu la même configuration tant pour l'entraînement que pour l'évaluation des modèles. L'entraînement a été réalisé sur les 1000 premières images de la base de données

1. <sup>‡</sup> voir **Remarque 6.3.2**.

2. <sup>†</sup> voir section 10.1 pour plus de détails sur l'implémentation.

MNIST, et l'évaluation a été effectuée sur les 1000 images suivantes. Cette méthode uniforme permet de garantir que les résultats sont comparables entre les différents modèles et configurations.

Les résultats suivants ont été obtenus pour les modèles VGG-16 Full, ResNet-50 et Vision Transformer (ViT).

Model	Mechanism	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
VGG-16 <sup>†</sup>	LC	771.56%	87.04%	99.5% / 99.5% (-0.0%)
	LC + dLoRA	6160.776%	98.378%	98.9% / 99.5% (-0.6%)
ResNet-50	LC	760.59%	86.85%	100.0% / 100.0% (-0.0%) <sup>‡</sup>
	LC + dLoRA	761.939%	86.876%	99.3% / 100.0% (-0.7%) <sup>‡</sup>
ViT-Tiny	LC	244.15%	59.04%	76.0% / 76.0% (-0.0%)
	LC + dLoRA	223.928%	55.343%	75.6% / 76.0% (-0.4%)
AlexNet	LC	813.32%	87.71%	98.2% / 99.0% (-0.8%)
	LC + dLoRA	28865.393%	99.654%	96.2% / 99.0% (-2.8%)
VGG-16*	LC	811.67%	87.68%	99.9% / 99.9% (-0.0%)
	LC + dLoRA	4562.891%	97.808%	99.1% / 99.9% (-0.8%)
LeNet	LC	562.26%	82.22%	96.1% / 96.1% (-0.0%)
	LC + dLoRA	1885.761%	94.697%	91.2% / 96.1% (-4.9%)

TABLE 7 – Résultats de compression sur MNIST avec VGG-16 Full, ResNet-50 et ViT-T<sup>‡</sup>

Le modèle VGG-16 Full Version, avec l'application de la technique de compression LC + dLoRA, montre une compression impressionnante avec une faible dégradation de la précision, ce qui suggère que cette méthode de compression est très efficace même pour des architectures de réseau profondes et complexes. Le ResNet-50, reconnu pour sa capacité à bénéficier de résidus pour améliorer l'apprentissage, a également montré des taux de compression favorables avec des pertes minimales en précision. Enfin, le Vision Transformer, un modèle basé sur des mécanismes d'attention plutôt que sur la convolution, a réussi à atteindre des taux significatifs avec des réductions acceptables en précision. Ces expériences confirment la robustesse de notre approche de compression à travers différents paradigmes d'architecture de modèles et soulignent l'importance de choisir judicieusement les configurations de compression pour maintenir un équilibre entre efficacité et précision.

Il est pertinent de noter que bien que le ResNet-50 affiche un taux de compression important, ce taux reste relativement modeste par rapport à des modèles plus

1. <sup>†</sup> VGG-16 Full Version

2. \* VGG-16 Lite Version

3. <sup>‡</sup> voir **Remarque 6.3.2.**

4. <sup>‡</sup> voir section 10.1 pour plus de détails sur l'implémentation.

anciens comme le VGG-16. Cette observation peut s'expliquer par la structure architecturale du ResNet-50 lui-même. Contrairement au VGG-16, qui contient plusieurs couches entièrement connectées (fully connected layers) qui sont généralement riches en paramètres, le ResNet-50 utilise une quantité réduite de telles couches. En effet, la majorité de l'architecture du ResNet-50 est composée de blocs résiduels qui ne contiennent qu'un petit nombre de couches entièrement connectées. Par conséquent, l'application de techniques de compression comme Delta-LoRA, qui ciblent principalement ces couches, n'entraîne pas une réduction aussi substantielle du nombre total de paramètres. Cette spécificité architecturale souligne l'importance d'adapter les stratégies de compression aux particularités de chaque modèle pour maximiser l'efficacité sans compromettre la précision des prédictions.

En ce qui concerne le Vision Transformer - Tiny (ViT-Tiny) mentionné dans le tableau, cette version allégée, a été choisie pour débuter nos expérimentations en raison de considérations de calcul, permettant ainsi de tester nos méthodes de compression sur des architectures basées sur des mécanismes d'attention tout en limitant les ressources requises. Le ViT-Tiny est configuré avec un nombre de têtes d'attention ('n\_heads') égal à 2, un nombre de blocs ('n\_blocks') de 2, une dimension cachée ('hidden\_dim') de 8 et une dimension pour les couches multi-perceptron ('mlp\_size') de 24. Étant donné que les couches linéaires dans les blocs de multi-head self attention ne contiennent pas autant de paramètres que les couches entièrement connectées des architectures plus traditionnelles, les gains de compression sur ces couches sont également limités. Cela illustre une fois de plus l'importance d'adapter les techniques de compression aux spécificités structurelles de chaque modèle pour optimiser à la fois la compression et la performance.

### 6.3.2 Vision Transformer-Small puis étude des couches d'application du Delta-LoRA

En raison de la compression insuffisante obtenue avec le ViT-Tiny sur MNIST, nous avons décidé de développer un modèle ViT-Small (ViT-S), qui est une configuration plus avancée et plus proche du plus petit modèle de ViT disponible dans le papier original. ViT-S est doté de huit têtes d'attention ('n\_heads'), huit blocs ('n\_blocks'), une dimension cachée ('hidden\_dim') de 512, et une dimension de couches multi-perceptron ('mlp\_size') de 2048. Cette configuration plus robuste vise à explorer le potentiel de compression sur un modèle qui intègre des caractéristiques structurelles plus complexes. De plus, dans le cadre de nos expérimentations avec le ViT-Small, nous avons évalué l'application de la technique Delta-LoRA non seulement sur les couches entièrement connectées mais aussi spécifiquement sur les composants de la multi-head self-attention (MHSA ou MSA), en particulier sur les couches 'query' et 'value' comme suggéré dans le papier de recherche de LoRA. Cette démarche visait à déterminer quelles couches bénéficient le plus de la compression Delta-LoRA, conduisant à des résultats notables documentés ci-dessous.

Model	Mechanism	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
ViT-S (MSAxMLP)	LC	764.484%	86.919%	97.3% / 97.3% (-0.0%)
	LC + dLoRA	12644.316%	99.209%	91.4% / 97.3% (-5.9%)
ViT-S (MSA)	LC	767.055%	86.963%	96.7% / 96.7% (-0.0%)
	LC + dLoRA	790.168%	87.344%	91.4% / 96.7% (-5.3%)
ViT-S (MLP)	LC	761.956%	86.876%	97.0% / 97.2% (-0.2%)
	LC + dLoRA	8215.592%	98.783%	96.7% / 97.2% (-0.5%)

TABLE 8 – Comparaison du taux de compression sur MNIST entre ViT-S (MSAxMLP), ViT-S (MSA) et ViT-S (MLP)

Le taux de compression est nettement plus élevé lorsque l'approche delta LoRA est appliquée sur MSAXMLP par rapport à son application sur MSA et sur MLP. Ceci est intuitif, car en ajoutant delta LoRA à MSAXMLP, on compresse un plus grand nombre de paramètres, augmentant ainsi l'efficacité de la compression. Le taux de compression de Delta-LoRA sur MSA et MLP se situe entre celui de l'application de delta LoRA sur MSA et celui sur MSAXMLP. Ce taux est plus proche de celui obtenu avec MSAXMLP, ce qui est logique puisque le nombre de paramètres dans MLP est plus significatif que dans MSA. En conclusion, delta LoRA démontre un effet additif : plus il y a de paramètres à compresser, plus le taux de compression est élevé.

**Remarque :** *Les résultats obtenus, en particulier le 100% d'accuracy pour le ResNet-50 sur MNIST et pour la version allégée de VGG-16 sur CIFAR-10, suggèrent une situation d'overfitting. Après vérification, il s'est avéré que notre test d'overfitting était confirmé : les images de l'ensemble d'entraînement et de l'ensemble d'évaluation étaient fortement similaires. Par conséquent, les valeurs obtenues doivent être interprétées avec prudence. Cette prise de conscience est intervenue un peu tardivement dans le processus. Dans la suite du stage, nous avons pris en compte cette problématique en nous focalisant sur l'implémentation de l'apprentissage incrémental. Le développement de cette nouvelle approche est détaillé dans la section suivante, où nous travaillons sur l'ensemble complet des données d'intérêt. Il est important de noter que le taux de compression est indépendant de l'accuracy, ce qui signifie que ces questions d'overfitting n'affectent pas nos mesures de compression.*

## 7 Empoisonnement des données

### 7.1 Motivation derrière l'application de l'apprentissage incrémental

L'implémentation de l'apprentissage incrémental dans notre schéma de points de sauvegarde est motivée par la nécessité de compléter le problème de *learning-to-defer*, où l'objectif est de permettre à l'IA de déléguer des décisions à des experts humains, malicieux ou non, lorsque sa précision est insuffisante. Cette stratégie est essentielle pour éviter les problèmes d'hallucination [24], qui sont fréquents dans les systèmes d'IA. Lorsqu'une délégation est effectuée, l'IA apprend de l'expertise fournie par l'humain. Par conséquent, il est crucial de former le modèle de manière à anticiper un comportement malicieux potentiel. Ainsi, si un tel comportement est détecté, l'entraînement peut être reprise juste avant cette détection.

Dans un contexte d'entraînement traditionnel de réseaux de neurones profonds, il est généralement admis que l'ensemble de données est correct par défaut. Le modèle apprend en parcourant cet ensemble à chaque époque et répète ce processus plusieurs fois jusqu'à ce qu'une condition d'arrêt soit atteinte : soit une stagnation de la précision de validation, soit une réduction insuffisante de l'erreur, ou le nombre d'époques prédéfini est atteint. Cependant, l'apprentissage incrémental propose une nouvelle approche dans le domaine de l'apprentissage profond. Ce mode d'apprentissage permet au modèle d'apprendre continuellement à partir des données qui arrivent, s'adaptant à chaque nouvel ensemble de données rencontré. Dans notre projet, nous nous concentrerons initialement sur des scénarios où chaque nouvel ensemble de données provient de la même source originale (telle que MNIST ou CIFAR-10) et contient toutes les classes de l'ensemble de données d'origine.

En intégrant l'empoisonnement des données dans notre schéma de points de sauvegarde, nous abordons un aspect crucial de la sécurité et de la robustesse des modèles d'apprentissage machine. Ce dispositif vise à garantir que le modèle reste fiable et performant, même en présence de données potentiellement corrompues, préservant ainsi l'intégrité de l'apprentissage face à des attaques potentielles.

### 7.2 Hypothèses sur le découpage de l'ensemble de données

Pour le découpage spécifique de l'ensemble de données destiné à l'entraînement, nous adoptons une stratégie qui garantit une diversité et une représentativité adéquates au sein de chaque sous-ensemble, tout en conservant l'intégrité de l'ensemble d'évaluation qui reste inchangé.

Le découpage s'articule de la manière suivante :

- **Quatre dataloaders** comportant **25%** de l'ensemble d'intérêt. Chaque dataloader est conçu pour s'assurer que les images sont différentes les unes des autres et que toutes les classes de l'ensemble d'intérêt sont représentées. Cette configuration permet un apprentissage profond et robuste à partir de segments substantiels de l'ensemble de données.
- **Vingt dataloaders** contenant chacun **5%** de l'ensemble d'intérêt. Ils garantissent que les images maintiennent une représentation complète des classes, augmentant ainsi la fréquence des mises à jour du modèle et la diversité des données vues par le modèle au cours d'une epoch.
- **Quatre-vingts dataloaders** incluant chacun **1.25%** de l'ensemble d'intérêt. Ces petits dataloaders offrent une granularité encore plus fine, idéale pour des tests rapides et des ajustements fréquents du modèle, permettant des ajustements précis et ciblés.

Un défi majeur dans l'apprentissage incrémental est le phénomène de *catastrophic forgetting*, terme qui décrit la tendance des modèles de réseaux de neurones à oublier les informations apprises antérieurement lorsqu'ils sont exposés à de nouvelles données. Ce concept est intuitif et naturel, analogiquement semblable au cerveau humain qui peut oublier des informations précédemment apprises lorsqu'il traite de nouvelles informations. Dans le cadre de l'apprentissage incrémental, chaque nouveau dataloader peut potentiellement éroder les connaissances acquises à partir des dataloaders précédents [26].

Cette structuration minutieuse est essentielle pour soutenir nos objectifs d'apprentissage incrémental, où chaque nouvelle portion de données doit enrichir le modèle sans réintroduire des biais précédemment corrigés.

## 7.3 Résultats avec l'apprentissage incrémental

### 7.3.1 Étude des configurations du découpage de l'ensemble d'entraînement d'intérêt

Pour notre étude, on a considéré les trois découpages mentionnés précédemment pour l'apprentissage incrémental et les résultats suivants ont été obtenus pour le modèle LeNet-5 sur MNIST (dont les configurations seront données après les résultats) :

Split	Methods	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
4*25% (15k-15k-15k-15k)	LC	888.872%	88.75%	98.87%/98.85% (+0.02%)
	LC + dLoRA	2481.857%	99.209%	92.09%/98.85% (-6.76%)
20*5% (20* 3k)	LC	723.688%	86.182%	98.39%/98.37% (+0.02%)
	LC + dLoRA	2270.412%	95.596%	92.63%/98.37% (-5.74%)
80* 1.25% (80* 750)	LC	885.524%	88.71%	97.65%/97.75% (-0.1%)
	LC + dLoRA	2478.693%	95.97%	94.08%/97.75% (-3,67%)

TABLE 9 – Comparaison du taux de compression sur MNIST pour LeNet-5 sur les trois configurations 4\*25%, 20\*5% et 80\*1.25%

Le taux de compression ne diffère pas tant que cela en fonction des configurations. Cependant, sur la configuration 80\*1.25%, la perte de la performance est moins importante que les autres. Le modèle apprend ainsi mieux en apprentissage incrémental lorsqu'il y a moins de données à apprendre.

Ensuite, une étude a également été menée pour le cas 4\*25% sur ViT-B/16 et les résultats suivants ont été obtenus :

Split	Methods	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
4*25% (4* 12.5k)	LC	591.368%	83.083%	98.70%/98.74% (-0.04%)
	LC + dLoRA	8173.368%	98.777%	97.79%/98.74% (-0.95%)

TABLE 10 – Taux de compression sur CIFAR-10 pour ViT-B/16 sur 4\*25%

Tous les résultats précédents ont été obtenus en utilisant les paramètres agrégés dans le tableau ci-dessous où l'utilisation de mêmes paramètres et hyperparamètres a été effectué afin de pouvoir comparer les résultats de manière juste et d'avoir des résultats comparables.

<b>Dataset Splitting</b>	4*15k (=25%)	20*3k (=5%)	80*750 (=1.25%)	4*12.5k (=25%)
<b>Model</b>	LeNet-5		ViT-B/16	
<b>Branching Point</b>	74.6%		HF <sup>†</sup> pretrained ImageNet1k	
<b>Dataset</b>	MNIST		CIFAR-10	
<b>Bit-width</b>	3		3	
<b>LoRA Scaling</b>	0.5		0.5	
<b>Batch Size</b>	128		128	
<b>Learning Rate</b>	0.08		4e-5	
<b>Epochs</b>	100		5	
<b>Super-Step</b>	Every 10 iterations		Every 10 iterations	
<b>Stopping Criterion</b>	Early stopping		Early stopping	
<b>Optimizer</b>	SGD		Adam <sup>¶</sup>	

TABLE 11 – Comparaison des configurations de LeNet-5 et ViT-B/16 sur les différents datasets

Plus d’informations concernant l’évolution de la performance au fur et à mesure des dataloader pendant l’apprentissage incrémental sont agrégés dans les tableaux :

Split	Methods	Compression ratio	Space Savings	Full Validation Accuracy (Restored / Full)
4* 25% (15k-15k-15k-15k)	LC + dLoRA	2481.857%	95.97%	88.74%-90.22%-91.33%-92.09%/98.85% (-6.76%)
	LC	888.872%	88.75%	98.11%-98.70%-98.83%-98.87%/98.85% (+0.02%)
	nothing	100.00%	0%	98.13%-98.73%-98.83%-98.85%
20* 5% (20* 3k)	LC + dLoRA	2270.412%	95.596%	84.36%-...-92.63%/98.37% (-5.74%)
	LC	723.688%	86.182%	93.81%-...-98.39%/98.37% (+0.02%)
	nothing	100.00%	0%	93.82%-...-98.37%
80* 1.25% (80* 750)	LC + dLoRA	2478.693%	95.97%	81.16%-...-94.08%/97.75% (-3.67%)
	LC	885.524%	88.71%	87.95%-...-97.65%/97.75% (-0.1%)
	nothing	100.00%	0%	88.00%-...-97.75%

FIGURE 7.1 – Evolution de la performance pour LeNet-5 pendant l’apprentissage incrémental pour les différentes configurations

4* 25% (4*12.5k)	LC + dLoRA	8173.368%	98.777%	97.79% / 98.74% (-0.95%)
	LC	591.368%	83.083%	98.70% / 98.74% (-0.04%)
	nothing	100.00%	0%	98.74%

FIGURE 7.2 – Evolution de la performance pour ViT-B/16 pendant l’apprentissage incrémental

1. <sup>†</sup> HF ou Hugging Face. Le point de branchement a été récupéré sur le site de Hugging Face
2. <sup>¶</sup> Adam ou Adaptive Moment Estimation [?] est une méthode de SGD, populaire pour sa capacité à s’adapter au problème spécifique en ajustant les taux d’apprentissage

### 7.3.2 Étude du rang en apprentissage incrémental

Le rang optimal pouvant évoluer d'un dataloader à l'autre, une analyse du rang a été considérée. Une première étude a été menée en utilisant un rang constant, soit le même rang pour chaque dataloader et des premiers résultats ont été obtenus pour Vision Transformer-B/16 pré-entraîné sur ImageNet-1k, adapté à l'ensemble d'entraînement CIFAR-10 :

Rank	Methods	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
16	LC	534.542%	81.292%	98.27%/98.23% (+0.04%)
	LC + dLoRA	7325.409%	98.63%	96.56%/98.23% (-1.67%)
4	LC	534.414%	81.29%	98.27%/98.21% (+0.06%)
	LC + dLoRA	8675.518%	98.847%	96.71%/98.21% (-1.5%)

TABLE 12 – Comparaison des taux de compression pour différents rangs dans la configuration 4\*25% avec rang = 16 et rang = 4

Résultats obtenus dans les conditions suivantes :

Rank	16	4
<b>Delta-LoRA Layers application</b>	MSA x MLP of the Transformer Encoder	
<b>Model</b>	ViT-B/16	
<b>Branching Point</b>	Hugging Face (pretrained on ImageNet-1k)	
<b>Dataset</b>	CIFAR-10	
<b>Dataset splitting</b>	25%-25%-25%-25%	
<b>Bit-width</b>	3	
<b>LoRA Scaling</b>	0.5	
<b>Batch Size</b>	128	
<b>Learning Rate</b>	4e-5	
<b>Epochs</b>	1	
<b>Super-Step</b>	Every 10 iterations	
<b>Stopping Criterion</b>	Early stopping	

TABLE 13 – Configuration pour la comparaison des taux de compressions pour rang = 16 et 4 dans la configuration 4\*25% pour ViT-B/16 sur CIFAR-10

Ensuite, observant que Vision Transformer-B/16 pré-entraîné sur ImageNet-1k obtient des performances similaires pour des rangs différents mais obtenant des taux de compression plus intéressant pour un rang faible, une seconde étude a été menée

pour approfondir ce rang faible dans le même contexte, et en utilisant plus de nombre d'epochs par ailleurs, obtenant :

Rank	Methods	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
4	LC	554.215%	81.956%	98.47%/98.5% (-0.03%)
	LC + dLoRA	8675.55%	98.847%	97.53%/98.5% (-0.91%)
3	LC	959.664%	89.58%	98.51%/98.51% (-0.00%)
	LC + dLoRA	8810.807%	98.865%	97.53%/98.51% (-0.92%)
2	LC	553.623%	81.937%	98.49%/98.49% (-0.00%)
	LC + dLoRA	8950.415%	98.883%	97.48%/98.49% (-1.01%)
1	LC	959.664%	89.58%	98.33%/98.51% (-0.18%)
	LC + dLoRA	9094.627%	98.9%	97.56%/98.51% (-0.95%)

TABLE 14 – Comparaison des taux de compression pour différents rangs dans la configuration 4\*25% pour ViT-B/16 pré-entraîné sur ImageNet-1k, adapté sur CIFAR-10

Ces résultats montrent que la performance est similaire pour les différents rangs = 1, 2, 3 et 4. La seule différence se fait au niveau du taux de compression, où un rang de 1 obtient le meilleur taux de compression, ce qui est intuitif puisqu'un rang de 1 indique moins d'informations ainsi plus de compression. Ainsi, un rang de 1 suffit pour maintenir une performance aussi bonne que les autres sur CIFAR-10, peut-être étant donné la simplicité de la tâche de classification de l'ensemble de donnée.

Etant donné le résultat précédent, il faut ainsi complexifier la tâche en utilisant un ensemble de donnée plus compliqué. Ainsi, l'ensemble de donnée Stanford Cars Dataset [30] a été suggéré, contenant 16 185 images et 196 classes. Par ailleurs, étant donné qu'on veut également observer la pertinence de notre schéma avec des modèles plus larges, on a décidé de faire la classification avec un Vision Transformer - Large (ViT-L/16 pré-entraîné sur ImageNet-21k) et les résultats suivants ont été obtenus :

Rank	Methods	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
8	LC	766.253%	86.949%	76.81%/77.19% (-0.38%)
	LC + dLoRA	7036.127%	98.579%	10.53%/77.19% (-66.66%)

TABLE 15 – Taux de compression et performance dans la configuration 4\*25% pour ViT-L/16 pré-entraîné sur ImageNet-21k, adapté sur Stanford Cars

Résultats obtenus dans les conditions suivantes :

<b>Rank</b>	8
<b>Delta-LoRA Layers application</b>	MSA x MLP of the Transformer Encoder
<b>Model</b>	ViT-L/16
<b>Branching Point</b>	Hugging Face (pretrained on ImageNet-21k)
<b>Dataset</b>	Stanford Cars
<b>Dataset splitting</b>	25%-25%-25%-25%
<b>Bit-width</b>	3
<b>LoRA Scaling</b>	0.5
<b>Batch Size</b>	64
<b>Learning Rate</b>	5e-5
<b>Learning Rate Delta-LoRA + LC</b>	7e-5
<b>Epochs</b>	3
<b>Super-Step</b>	Every 10 iterations
<b>Optimizer</b>	AdamW

TABLE 16 – Configuration pour le taux de compression pour rang = 8 dans la configuration 4\*25% de ViT-L/16 sur Stanford Cars

La performance du modèle suivant notre schéma Delta-LoRA + LC admet une perte de performance significative par rapport au modèle sans considérer notre schéma. Cela peut être dû à plusieurs facteurs, le premier étant la quantité d'images par classes utilisé pour l'adaptation (fine-tuning) et le second étant le nombre d'epoch nécessaire au fine-tuning. Ces résultats ont été obtenus en utilisant le serveur du NSCC, en se connectant au superordinateur ASPIRE 2A de Singapour, et cela a pris 1 jour et 16 heures, notamment dûe au temps nécessaire pour calculer la performance de chaque modèle dans notre boucle d'apprentissage sur l'ensemble de validation.

Pour le fine-tuning, les hyperparamètres ainsi que le choix de l'optimiseur (AdamW [31]) ont été finement ajustés afin de maintenir une accuracy de validation autour de 87%, comme le montre l'évolution de l'accuracy et de la loss dans les FIGURES 7.3 et 7.4.

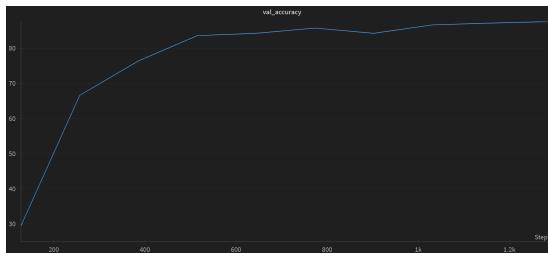


FIGURE 7.3 – Validation accuracy de ViT-L pré-entraîné sur ImageNet-21k, adapté à Stanford Cars

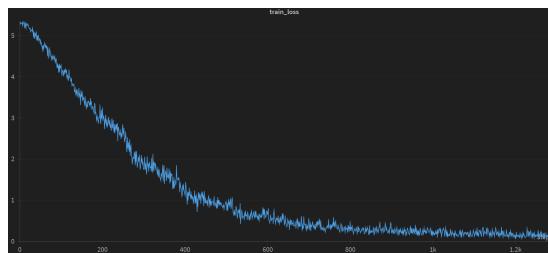


FIGURE 7.4 – Train Loss de ViT-L pré-entraîné sur ImageNet-21k, adapté à Stanford Cars

Néanmoins, il est à noter que la configuration utilisant le Delta LoRA + LC nécessitait plus d'époches pour atteindre une accuracy comparable à celle d'une configuration sans Delta LoRA + LC, ce qui engendre une augmentation considérable du temps d'entraînement que nous n'avons pas par manque de temps. Par ailleurs, le choix d'appliquer Delta-LoRA aux couches MSA et MLP de l'encodeur du transformer, motivée par l'envie d'obtenir un taux de compression important, cependant, la diminution de la performance peut justement être dûe par cet excès de compression, ainsi peut-être qu'une application uniquement sur les couches MSA de l'encodeur du transformer pourrait être plus pertinent.

Dans la continuité de cette étude sur l'apprentissage incrémental avec les rangs variables, nous avons adapté un modèle Vision Transformer-Large (ViT-L/16) pré-entraîné sur ImageNet-21k pour classifier le Stanford Cars Dataset, considérant une optimisation possible pour la séparation des dataloaders. En effet, avec un ensemble d'entraînement de 8144 images réparties sur 196 classes, cela nous donne en moyenne 42 images par classe. En divisant cet ensemble en quatre dataloaders de taille égale, chaque dataloader contiendrait environ une dizaine d'images par classe, ce qui pourrait être considéré comme insuffisant pour un apprentissage efficace. Pour pallier ce problème, une seconde étude a été entreprise où le premier dataloader englobe 70% de l'ensemble d'entraînement (garantissant au moins une image de chaque classe), et les 30% restants sont répartis équitablement entre les trois dataloaders suivants, soit 10% chacun. Cette configuration pourrait potentiellement permettre au modèle suivant notre schéma delta LoRA + LC de bénéficier de suffisamment de données pour un apprentissage efficace, atteindre une bonne accuracy initiale, et continuer à améliorer ses performances avec chaque nouveau dataloader.

Cette approche cherche à maximiser les bénéfices de l'apprentissage incrémental, tout en ajustant la granularité des données traitées pour optimiser les performances du modèle tout au long du processus de fine-tuning. Cependant, dans un soucis de temps, cette étude n'a pas pu donner de résultats concluant et une étude plus approfondie pourrait être conduite.

## 8 Vérification dans les ensembles de données

Dans tout processus d’entraînement de modèles d’apprentissage profond, la qualité des données utilisées est cruciale pour garantir la généralisation et la robustesse du modèle. Un aspect fondamental est la vérification préalable de la diversité et de la représentativité des images présentes dans les ensembles d’entraînement et de test.

### 8.1 Vérification des images uniques

Avant de procéder à l’entraînement, il est essentiel de s’assurer que les images dans l’ensemble d’entraînement sont distinctes de celles présentes dans l’ensemble de test. Cette vérification empêche le modèle de simplement mémoriser les images vues pendant l’entraînement, un phénomène qui conduit à l’overfitting, rendant ainsi le modèle incapable de généraliser à de nouvelles données. Une inspection manuelle des images peut être employé pour cette tâche (cf. FIGURE 8.1), garantissant ainsi que chaque image présente dans l’ensemble de test est réellement inconnue du modèle. Comme mentionné précédemment, l’overfitting peut fausser les résultats obtenus 6.3.2, et cette prise de conscience a motivé des ajustements dans la méthodologie employée par la suite.

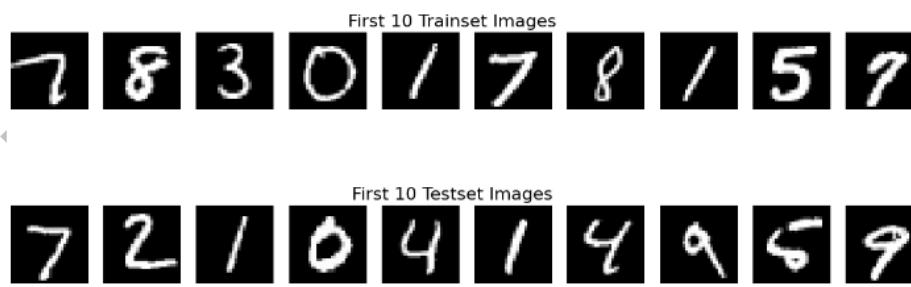


FIGURE 8.1 – Vérification de la différence des images entre l’ensemble d’entraînement et l’ensemble de test pour MNIST

Comme illustré dans la Figure 8.1, il est possible de constater que les dix premières images sont différentes entre l’ensemble d’entraînement et l’ensemble de test. Cette distinction visuelle est cruciale pour garantir l’absence de surapprentissage (*overfitting*) et assure que le modèle peut généraliser efficacement à de nouvelles données. La vérification que les ensembles de données ne contiennent pas d’images identiques constitue une étape essentielle pour évaluer la généralisabilité du modèle, renforçant ainsi la robustesse des résultats obtenus.

## 8.2 Vérification de la présence de toutes les classes dans les dataloaders

Dans le contexte de notre approche d'apprentissage incrémental, où plusieurs découpages de l'ensemble d'entraînement sont envisagés, l'hypothèse est que chaque dataloader doit être soigneusement conçu pour inclure toutes les classes présentes dans l'ensemble d'intérêt.

Pour chaque dataloader, un test est effectué pour s'assurer que toutes les classes sont bien représentées (cf. FIGURE 8.2). Cette vérification permet de maintenir la diversité des données vues par le modèle à chaque étape de l'entraînement, et ainsi de maximiser sa capacité de généralisation.

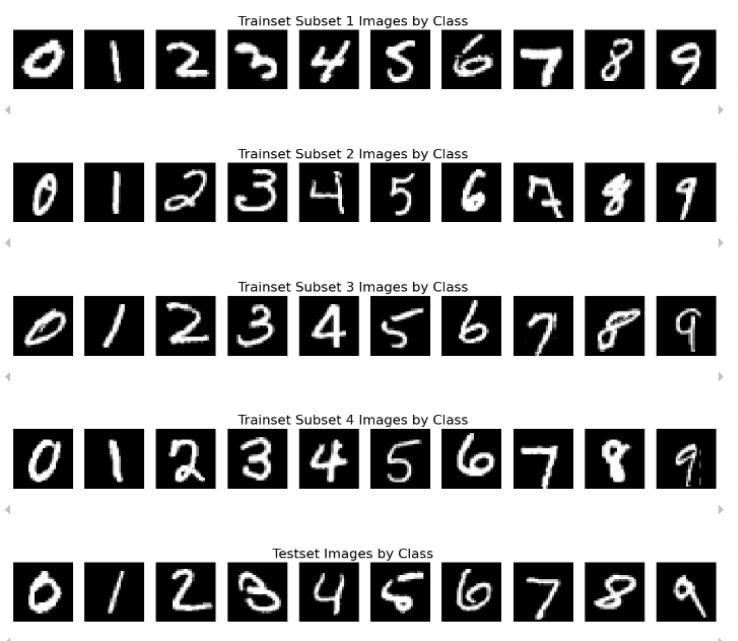


FIGURE 8.2 – Vérification de la présence de toutes les classes dans chaque dataloader pour MNIST

En ce qui concerne également l'ensemble de données Stanford Cars, une analyse de la présence des classes dans les dataloaders a également été effectuée (cf. FIGURE 8.3, FIGURE 8.4, FIGURE 8.5 et FIGURE 8.6).

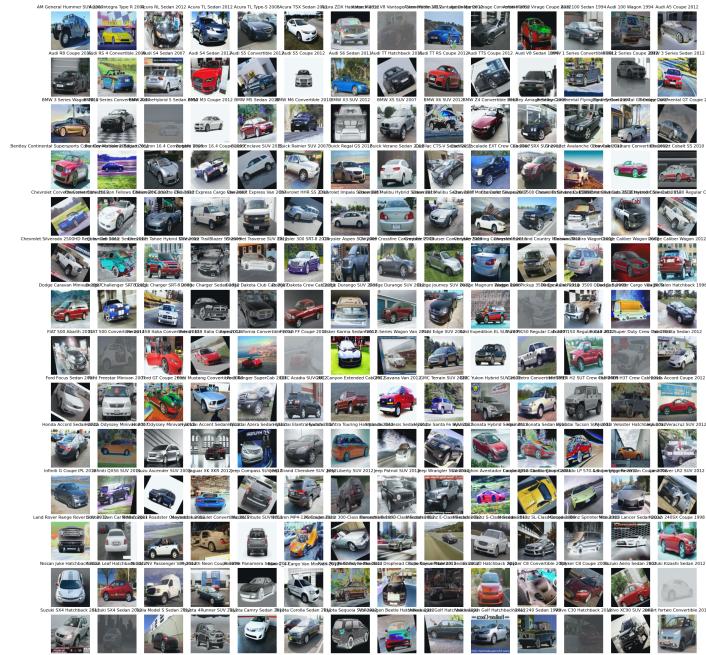


FIGURE 8.3 – Vérification de la présence de toutes les classes dans le dataloader 1 pour Stanford Cars

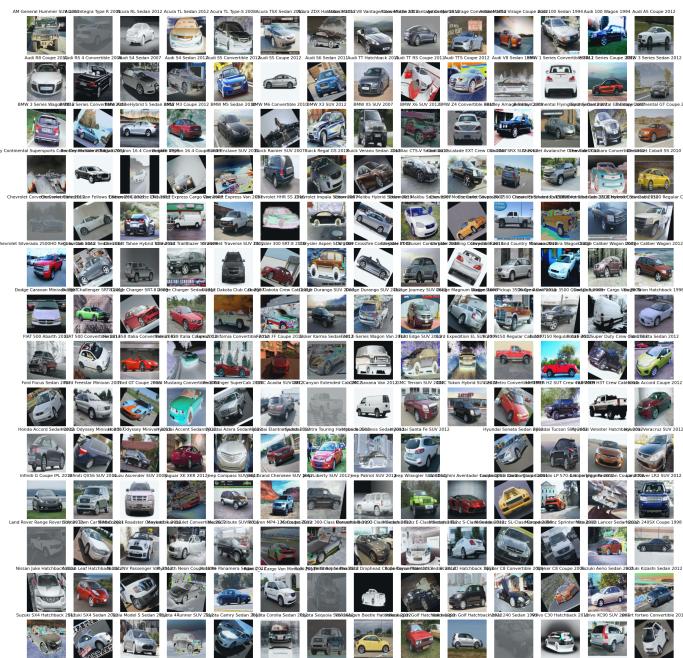


FIGURE 8.4 – Vérification de la présence de toutes les classes dans le dataloader 2 pour Stanford Cars

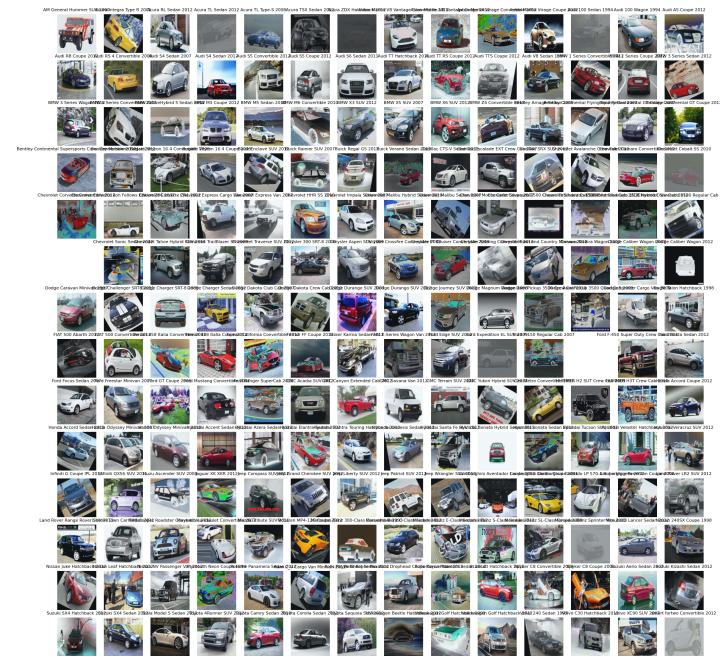


FIGURE 8.5 – Vérification de la présence de toutes les classes dans le dataloader 3 pour Stanford Cars

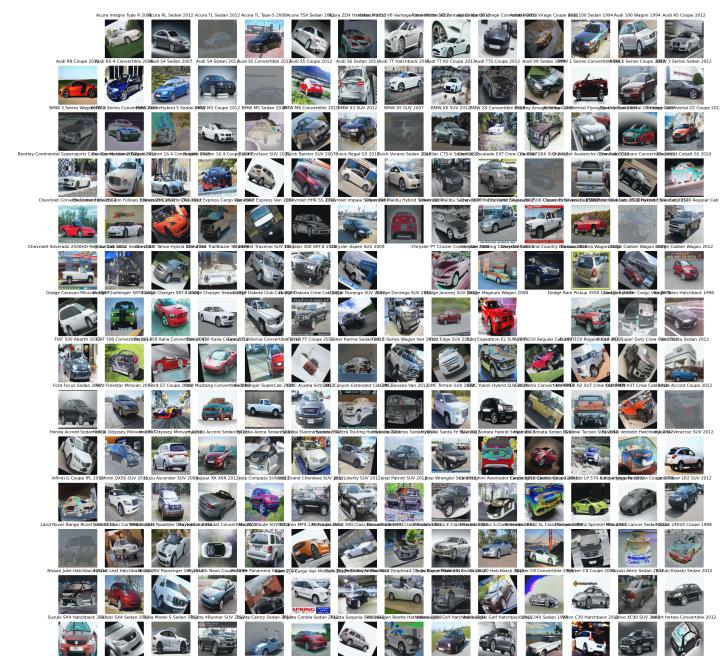


FIGURE 8.6 – Vérification de la présence de toutes les classes dans le dataloader 4 pour Stanford Cars

### 8.3 Synthèse

En intégrant ces vérifications préalables, le modèle est formé dans des conditions optimales, minimisant les risques d'overfitting et maximisant sa capacité à généraliser. Ces étapes critiques permettent de renforcer la confiance dans les résultats obtenus et de garantir la robustesse du modèle face à des données nouvelles.

## 9 Explication des choix des modèles

En se concentrant que sur la classification d'image pour ce projet, la motivation derrière le choix de nos modèles s'appuyait à la fois sur les performances et l'utilisation importants de ces derniers en classification d'image. Par ailleurs, on a également suivi des modèles qui suivaient l'ordre d'apparition des modèles, comme le montre la FIGURE 9.1 [25], où les modèles répondent à chaque fois aux limites rencontrées par les modèles précédents.

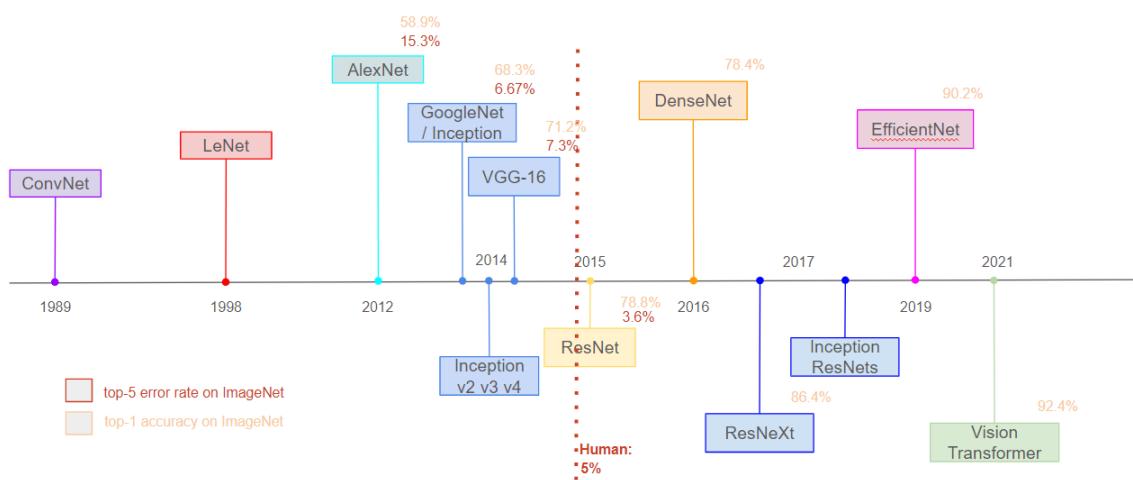


FIGURE 9.1 – Chronologie des modèles de classification d'images

## 10 Pseudo-code

Les codes sont disponibles sur le lien suivant : [https://github.com/BryanBradfo/pfe\\_lc\\_lora](https://github.com/BryanBradfo/pfe_lc_lora). Ci-dessous les pseudo-code expliquant l'implémentation du schéma de checkpointing efficace à la fois sans considérer l'apprentissage incrémental et ensuite considérant l'empoisonnement de données.

## 10.1 Checkpointing Efficace : Sans apprentissage incrémental

---

**Algorithm 1** Pseudo-code du schéma proposé sur un ViT

- 1: Initialiser les couches à décomposer, le rang pour la décomposition Delta-LoRA
  - 2: Initialiser les taux d'apprentissage du modèle sans et avec l'utilisation du schéma
  - 3: Charger le modèle pré-entraîné ViT pour la classification d'images
  - 4: Initialiser les optimiseurs pour chaque modèle
  - 5: Initialiser les itérateurs et les sets pour le suivi de l'entraînement
  - 6: **for** chaque batch dans l'ensemble d'apprentissage **do**
  - 7:     Entraîner les modèles avec et sans l'utilisation du schéma proposé
  - 8:     Enregistrer les états des modèles à des intervalles spécifiés
  - 9:     Evaluer la précision des modèles à des intervalles spécifiés
  - 10: **end for**
- 

## 10.2 Checkpointing Efficace : Avec apprentissage incrémental

---

**Algorithm 2** Pseudo-code du schéma proposé sur un ViT

- 1: Initialiser les couches à décomposer, le rang pour la décomposition Delta-LoRA
  - 2: Initialiser les taux d'apprentissage du modèle sans et avec l'utilisation du schéma
  - 3: Charger le modèle pré-entraîné ViT pour la classification d'images
  - 4: Initialiser les optimiseurs pour chaque modèle
  - 5: Définir les listes pour enregistrer les précisions
  - 6: Initialiser les itérateurs et les sets pour le suivi de l'entraînement
  - 7: **for** chaque lot de données dans l'ensemble d'apprentissage **do**
  - 8:     **for** chaque batch dans le lot de données **do**
  - 9:         Entraîner les modèles avec et sans l'utilisation du schéma proposé
  - 10:         Enregistrer les états des modèles à des intervalles spécifiés
  - 11:         Evaluer la précision des modèles à des intervalles spécifiés
  - 12:         Appliquer l'arrêt prématué (early stopping) basé sur l'amélioration de la précision
  - 13:     **end for**
  - 14:     Afficher les résultats
  - 15:     Réinitialiser les configurations pour le prochain lot
  - 16: **end for**
-

# 11 Conclusion

## 11.1 Synthèse des résultats et perspectives

L'intérêt d'un schéma de points de sauvegarde innovant a été démontré le long de ce rapport, exploitant les potentialités de LC-checkpoint et de Delta-LoRA. Des résultats ont été obtenus pour plusieurs architectures de réseaux de neurones profonds, notamment LeNet-5, AlexNet, VGG-16 lite version, VGG-16 full version, et Vision Transformer. Ces travaux ont été réalisés sur les bases de données CIFAR-10 et MNIST, et il est envisageable de les étendre à d'autres ensembles tels qu'ImageNet-1k ou à d'autres domaines tels que la compréhension du langage naturel (Natural Language Understanding), avec l'exploration de modèles tels que DistilBert, RoBERTa, et gpt2 sur des bases comme IMDb.

Le choix de découpage de la base de données pour l'apprentissage incrémental, ainsi que le choix du rang pour Delta-LoRA, pourraient être critiqués. Des recherches ultérieures pourraient envisager l'adoption d'un rang croissant avec l'accumulation de nouvelles données, permettant potentiellement de pallier la réduction de l'accuracy face à une complexité croissante des connaissances à intégrer.

Les travaux réalisés, bien que préliminaires pour une publication scientifique, ouvrent des pistes de recherche prometteuses. La réimplémentation plus approfondie des techniques LC-checkpoint et Delta-LoRA est nécessaire, leur code n'ayant pas été publié précédemment, et seule une vérification superficielle a été réalisée jusqu'à présent.

Des efforts supplémentaires pourraient être consacrés à la compression des données, ce qui optimiserait le temps de calcul en favorisant des opérations matricielles plus rapides en Python, plutôt que des calculs point par point. Ces améliorations augmenteraient significativement l'efficacité des méthodes employées.

En outre, une critique potentielle à l'égard de ce schéma pourrait être le choix des algorithmes utilisés. Si LC-checkpoint était considéré comme l'état de l'art en 2020, de nouveaux développements tels que QD-Compressor [27] en 2023, DynaQuant [28] en 2023 et ExCP [29] en 2024, ont depuis émergé. Il serait donc pertinent d'envisager l'adaptation de notre schéma en intégrant ces nouvelles méthodes de compression de checkpoints. Cependant, l'incorporation de ces technologies nécessiterait une recherche approfondie. De même pour le choix de Delta-LoRA, introduit en 2023, qui pourrait également bénéficier d'une évaluation comparative avec d'autres innovations récentes dans le domaine.

## 11.2 Compétences techniques développées

Ce stage a été particulièrement transformateur d'un point de vue technique, intégrant et approfondissant de nombreuses connaissances acquises à l'ENSEEIHT. Il m'a ouvert les portes du monde passionnant de la recherche en intelligence artificielle, m'exposant aux méthodes de travail rigoureuses requises dans ce domaine. Au cours de ce stage, j'ai découvert des techniques avancées telles que LoRA - Low Rank Adaptation, la quantification (quantization), ainsi que LC-checkpoint, qui incorpore diverses méthodes de compression des données. J'ai également eu l'occasion d'implémenter et d'expérimenter avec des modèles de classification d'images renommés tels que LeNet-5, AlexNet, VGG-16, ResNet-50, et Vision Transformer (ViT), tout en explorant en profondeur le mécanisme de self-attention au sein du modèle ViT.

L'utilisation de plateformes telles que Hugging Face a facilité l'accès et la manipulation de modèles open source, tandis que les outils tels que FileZilla et PuTTY ont été essentiels pour la gestion des fichiers et des tâches sur les superordinateurs ASPIRE 2A de Singapour via NSCC. De plus, j'ai pu analyser en détail le comportement des modèles lors du fine-tuning en utilisant des outils de visualisation comme Weight and Bias (wandb).

Ce stage a donc non seulement enrichi ma compréhension technique et ma connaissance des technologies de pointe en IA, mais a également renforcé ma capacité à travailler dans un environnement de recherche exigeant, tout en cultivant des relations professionnelles enrichissantes. Ces expériences m'ont confirmé l'importance de l'innovation continue et de la collaboration dans le domaine de la recherche en intelligence artificielle.

## Références

- [1] Dominique Baillargeat, *CNRS@Create*. CNRS - Centre national de la recherche scientifique, 2019.
- [2] Heng Swee Keat, *National Research Foundation*. Deputy Prime Minister and Coordinating Minister for Economic Policies, 2006.
- [3] Yu Chen, Zhenming Liu, Bin Ren, Xin Jin, *On Efficient Constructions of Checkpoints*. arXiv :2009.13003, 2020. [cs.LG]
- [4] Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, Lei Zhang, *Delta-LoRA : Fine-Tuning High-Rank Parameters with the Delta of Low-Rank Matrices*. arXiv :2309.02411, 2023. [cs.LG]
- [5] Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner, *Gradient Based Learning Applied to Document Recognition*. Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, 1998.

- [6] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems 25 (NeurIPS 2012), University of Toronto, 2012.
- [7] Karen Simonyan, Andrew Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv preprint arXiv :1409.1556, Oxford University, 2014.
- [8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition*. arXiv :1512.03385, Microsoft, 2015.
- [9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby, *An Image is Worth 16x16 Words : Transformers for Image Recognition at Scale*. arXiv :2010.11929, Google, 2020.
- [10] Yann LeCun, Corinna Cortes, Christopher J.C. Burges, *Modified National Institute of Standards and Technology database*. 1994.
- [11] Alex Krizhevsky, Vinod Nair, Geoffrey Hinton, *Canadian Institute For Advanced Research - 10*. 2009.
- [12] Alex Krizhevsky, Vinod Nair, Geoffrey Hinton, *Canadian Institute For Advanced Research - 100*. 2009.
- [13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*. arXiv :1409.0575, Stanford, 2013.
- [14] Chen Y., Liu Z., Ren B., Xin J., *On Efficient Constructions of Checkpoints*. arXiv :2009.13003, 2020.
- [15] Goodfellow I., Bengio Y., and Courville A., *Deep learning*. MIT press, 2016
- [16] Ioffe S. and Szegedy C., *Batch normalization : Accelerating deep network training by reducing internal covariate shift*. arXiv :1502.03167, 2015.
- [17] Hu E., Shen Y., Wallis P., Allen-Zhu Z., Li Y., Wang L., Chen W., *LoRA : Low-Rank Adaptation of Large Language Models*. arXiv :2106.09685, 2021.
- [18] Li C., Farkhoor H., Liu R., Yosinski J., *Measuring the Intrinsic Dimension of Objective Landscapes*. arXiv :1804.08838, 2018.
- [19] Niederfahrenhorst A., Hakhamaneshi K., Ahmad R., *Fine-Tuning LLMs : LoRA or Full-Parameter ? An in-depth Analysis with Llama 2*. anyscale, 2023.
- [20] Zi B., Qi X., Wang L., Wang J., Wong K., Zhang L., *Delta-LoRA : Fine-Tuning High-Rank Parameters with the Delta of Low-Rank Matrices*. arXiv :2306.10925, 2023.
- [21] David A. Huffman, *A Method for the Construction of Minimum-Redundancy Codes*. Proceedings of the IRE, Vol. 40, No.9, pp. 1098-1101, Sept. 1952.

- [22] Ziv J., Lempel A., *A Universal Algorithm for Sequential Data Compression.* IEEE Transactions on information theory, Vol. it-23, No.3, May 1977.
- [23] Robbins H., Monro S., *A Stochastic Approximation Method.* University of North Carolina, 1951.
- [24] Maynez J., Narayan S., Bohnet B., McDonald R., *On Faithfulness and Factuality in Abstractive Summarization.* arXiv :2005.00661, Google Research, May 2020.
- [25] Pragati Baheti, *A Comprehensive Guide to Convolutional Neural Networks.* V7Labs, Microsoft, 2021.
- [26] McCloskey M., J. Cohen N., *Catastrophic Interference in Connectionist Networks : The Sequential Learning Problem.* Academic Press, Psychology of Learning and Motivation, vol. 24, pp. 109-165, 1989.
- [27] Jin H., Wu D., Zhang S., Zou X., Jin S., Tao D., Liao Q., Xia W., *Design of a Quantization-based DNN Delta Compression Framework for Model Snapshots and Federated Learning.* Washington State University, EECS, 2023.
- [28] Agrawal A., Reddy S., Bhattacharya S., Prabhakara Sarath Nookala V., Vaishishth V., Rong K., Tumanov A., *DynaQuant : Compressing Deep Learning Training Checkpoints via Dynamic Quantization.* arXiv :2306.11800, Georgia Institute Of Technology, University of Oxford, 2023.
- [29] Li W., Chen X., Shu H., Tang Y., Wang Y., *ExCP : Extreme LLM Checkpoint Compression via Weight-Momentum Joint Shrinking.* arXiv :2406.11257, Huawei, 2024.
- [30] Krause J., Jin H., Yang J., Fei-Fei L., *Fine-Grained Recognition without Part Annotations.* arXiv :1702.01721, Adobe Research, Stanford, 2013
- [31] Loshchilov I., Hutter F., *Decoupled Weight Decay Regularization.* arXiv :1711.05101, ICLR, 2019.