Vers des algorithmes d'apprentissage automatique économes en E/S pour les systèmes embarqués : application aux K-means et Random Forests

Camélia Slimani

THESE DE DOCTORAT DE

L'UNIVERSITE
DE BRETAGNE OCCIDENTALE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Camélia SLIMANI

Vers des algorithmes d'apprentissage automatique économes en E/S pour les systèmes embarqués : application aux K-*means* et *Random Forests*

Thèse présentée et soutenue à Brest, le 11 Juillet 2022
Unité de recherche : Lab-STICC UMR CNRS 6285

Rapporteurs avant soutenance :

Gilles SASSATELLI	Directeur de Recherche CNRS au LIRMM de Montpellier
François TRAHAY	Maître de Conférences Habilité à Diriger des Recherches à Télécom Sud Paris

Composition du Jury :

Président :	Frank SINGHOFF,	Professeur à l'Université de Bretagne Occidentale
Examinateurs :	Kaoutar EL MAGHRAOUI,	Docteure, IBM US
	Smail NIAR,	Professeur à l'Université Polytechnique Hauts-de-France
	Gilles SASSATELLI,	Directeur de Recherche CNRS au LIRMM de Montpellier
	François TRAHAY,	Maître de conférences Habilité à Diriger des Recherches à Télécom Sud Paris
Dir. de thèse :	Jalil BOUKHOBZA	Professeur à ENSTA Bretagne
Enc. de thèse :	Stéphane RUBINI	Maître de Conférences à l'Université de Bretagne Occidentale

Résumé

L'abondance des données numériques collectées offre l'opportunité d'alimenter des modèles intelligents, capables d'extraire de la connaissance exploitable par l'humain. L'entraînement de ces modèles, par des algorithmes de *Machine Learning* (ML), requiert d'importantes ressources en puissance de calcul, en mémorisation et en stockage. Par conséquent, cette tâche a longtemps été déléguée à des calculateurs puissants. Néanmoins, cela nécessite le transfert des données des dispositifs de collecte, vers les calculateurs, ce qui expose les données collectées à des problématiques de sécurité et de confidentialité d'une part, et augmente les coûts énergétiques et temporels dus aux transferts d'autre part. Dans ce contexte, pour s'affranchir des coûts de communications, le paradigme de l'*Edge Intelligence* émerge, et rapproche l'« intelligence » au plus près des dispositifs de collecte.

L'*Edge Intelligence* soulève, néanmoins, de nombreux défis scientifiques, notamment l'exécution des algorithmes de ML sous des contraintes temporelles, énergétiques, et d'espace mémoire, présentes sur les dispositifs embarqués. Dans le cadre de cette thèse, nous nous intéressons particulièrement à la contrainte d'espace en mémoire principale. En effet, cet espace sur les dispositifs de collecte est limité et peut, dans certains cas, être insuffisant pour contenir les données d'apprentissage. Lorsque le volume de données à traiter est supérieur à l'espace mémoire disponible, le système d'exploitation utilise le mécanisme de *swap* afin d'étendre l'espace mémoire disponible par une partie du stockage secondaire. Par conséquent, l'exécution de la phase d'apprentissage est ralentie par les accès au stockage secondaire (E/S), bien plus lents que ceux à la mémoire principale. Notre objectif est d'analyser le motif des E/S des algorithmes de ML, de déterminer son origine, et de proposer une méthode de réduction des E/S pour accélérer ces algorithmes sur des platformes embarquées.

Dans cette thèse, nous nous sommes intéressés à deux cas d'études algorithmiques qui sont : le K-MEANS et les RANDOM FORESTS, pour lesquels nous avons mesuré que respectivement, 70% et 90% des temps d'exécution étaient consacrés aux E/S lorsque les volumes des données d'apprentissage étaient 4 fois supérieurs à l'espace mémoire disponible. Pour ces deux algorithmes, nous avons analysé et identifié l'origine algorithmique des E/S. Pour le *K-means*, nous avons mis en évidence la corrélation entre vitesse de convergence de l'algorithme et nombre de parcours complets des données d'apprentissage. La solution que nous proposons, alors, consiste à appliquer l'approche « diviser pour régner », en appliquant le *K-means* sur de petites parties de données, suivi d'une méthode d'agrégation du résultat final. Nous réduisons, ainsi, le temps d'exécution du *K-means* de 60% en moyenne. Pour les *Random Forests*, nous avons identifié la faible localité spatiale et l'accès à des données inutiles comme causes algorithmiques de l'important volume d'E/S. La solution proposée est fondée sur deux mécanismes : le premier consiste à ré-organiser les données de manière à améliorer la localité spatiale, et le deuxième consiste à charger les données, à la demande, autrement dit, uniquement lorsqu'elles doivent être accédées. De cette manière, nous réduisons le temps de construction d'une *Random Forest* de 70% en moyenne.

En résumé, dans cette thèse, nous avons montré à travers deux exemples, que la révision des algorithmes permet de faire face à l'augmentation du volume de données

d'apprentissage en réduisant le volume des E/S. L'extension de cette démarche à d'autres algorithmes/familles d'algorithmes de ML pourrait faire l'objet de travaux futurs.

Mots clés : *Edge Intelligence*, *Machine Learning*, contrainte mémoire, E/S, *K-means*, *Random Forests*.

Abstract

The abundance of collected digital data offers an opportunity to feed intelligent models, capable of extracting knowledge that can be exploited by humans. Training these models, by Machine Learning (ML) algorithms, requires significant resources in terms computing power, memory and storage. Therefore, this task has been generally delegated to powerful computers. Nevertheless, this requires data transfer from the collecting devices, to these computers, which exposes collected data to security and privacy issues on the one hand, and increases the energy and time costs due to network transfers on the other hand. In this context, in order to avoid communication costs, the Edge Intelligence paradigm emerged, and brings intelligence closer to the data collection devices.

Edge Intelligence raises many scientific challenges, especially the execution of ML algorithms under temporal, energy, and memory space constraints specific to embedded devices. In this thesis, we are particularly interested in the memory constraint issue. Indeed, the available main memory space on the collecting devices is limited and may, in some cases, be insufficient to contain all the training data. When the volume of data to be processed is larger than the available memory space, the operating system leverages the *swap* mechanism in order to extend the memory space using the secondary storage. Consequently, the execution of the learning phase is dramatically slowed down by the generated I/Os to the secondary storage. Our objective is to analyze the I/O pattern of ML algorithms, to determine its origin, and to propose a method for I/Os reduction to accelerate these algorithms on embedded devices.

In this thesis, we are interested in two algorithmic case studies : K-MEANS and RANDOM FORESTS, for which we have shown that respectively, 70% and 90% of the overall execution time was dedicated to I/Os when the data-sets sizes were 4 times larger than the available memory space. For these two algorithms, we analyzed and identified the algorithmic origin of the I/Os. For the *K*-means, we have highlighted the correlation between the convergence speed of the algorithm and the number of complete spans of the training data-set. The solution we propose, then, consists in making use of the divide-and-conquer approach, by applying the *K*-means on small chunks of the data-set, followed by a method of aggregation for the final result. Doing so, we reduced the execution time of *K*-means by 60% on average. For the Random Forests, we have identified the poor spatial locality and the frequent access to useless data as algorithmic causes of the high I/O volume. The proposed solution is based on two mechanisms : the first one consists in reorganizing the data in order to improve the spatial locality, and the second one consists in loading data on demand, i.e., only when they have to be accessed. In this way, we reduced the construction time of a Random Forest by 70% on average.

In this thesis, we have shown through two examples, that the revision of algorithms can cope with the increase in the volume of training data by drastically reducing I/Os. The extension of this approach to other ML algorithms/families of algorithms could be the subject of future research.

Key words : Edge Intelligence, Machine Learning, memory constraint, I/Os, *K*-means, Random Forests.

Table des matières

Liste des figures

Liste des tableaux

Acronymes

K-**NN** K-Nearest Neighbors. 44, 47, 48, 62

ANN Artificial Neural network. 47

ARI Adjusted Rand Index. 109–111

ASIC Application-Specific Integrated Circuit. 63, 64

BFS Breadth-First-Search. 73, 75, 120, 124

CLARA Clustering LARge Applications. 39

CLARANS Clustering LARge Applications based on Randomized Search. 39

CPU Core Process Unit. 55–57, 73, 89, 139, 140

DFS Depth-First-Search. 73, 75, 116, 120, 124

DRAM Dynamic Random Access Memory. 21, 22, 24, 56–59, 63, 101, 134

DT Decision Tree. 47, 48

E/S Entrées/Sorties. 2, 3, 5, 6, 9, 11, 15, 17, 24–27, 61, 65, 67, 75–81, 85, 87–89, 91, 94–101, 103–107, 109, 111, 112, 114, 116, 117, 119, 121, 122, 124, 128–132, 134–136, 139, 140, 142, 143

EDP Energy Delay Product. 141

EI Edge Intelligence. 2–5, 9, 11, 17–21, 23, 24, 27, 29–32, 34, 35, 40, 42, 52–55, 62, 63, 65, 72, 76, 97–99, 133, 134, 139, 140, 143

EM Expectation Maximization. 142

FeRAM Ferroelectric Random Access Memory. 58, 59

GMM Gaussian Mixture Model. 39, 111, 142

GPU Graphical Process Unit. 139, 140

HDD Hard Disk Drive. 58, 59

IDC International Data Corporation. 17

IoT Internet of Things. 17, 18, 23, 31

K-MLIO *K*-Means with a Low I/O. 6, 9, 11, 78, 81–85, 87–98, 135, 137–142, 148–150

K-MLIOR *K*-Means with a Low I/O with Randomization. 9, 89–97, 137, 138

ML Machine Learning. 2–4, 18, 21, 23, 24, 29–31, 36, 40, 52–54, 59, 60, 62, 63, 65, 76, 77, 97, 122, 134, 138, 139, 142, 143

MLaaS Machine Learning as a Service. 18, 22

MRAM Magnetic Random Access Mmeory. 58

NAS Neural Architecture Search. 62, 64

NBC Naive Bayes Classifier. 44, 47

NN Neural Network. 45

NVM Non-Volatile Memory. 56, 58, 59, 63, 74, 75

PAM Partitionning Around Medoids. 39

PCM Phase Change Memory. 58, 59

PRFA Page Frame Reclaiming Algorithm. 61

RaFIO Random Forest I/O-aware algorithm. 6, 9, 100, 106, 108, 117, 119, 120, 122, 126, 131, 132, 135, 137, 139–142, 149, 150

ReRAM Resistive Random Access Memory. 58, 59, 63

RF Random Forests. 2–6, 17, 26, 27, 29, 30, 46, 48, 53, 62, 65, 72, 74–76, 99–101, 105, 117, 119, 120, 126, 128, 130, 134–137, 143

RI Random Index. 109, 110

SAMM Systèmes Autonomes en Milieu Marin. 5, 17, 22, 23

SRAM Static Random Access Memory. 55–59

SSD Solid State Disk. 63, 96

SVM Support Vector Machine. 44, 47, 48, 142

XGBoost eXtreme Gradient Boosting. 142

Nomenclature

I/O_{cost} Coût en E/S de l'algorithme *K-means*

μ_i Centre du *cluster i*

$\vec{\mathrm{M}}_i$ Centre réel du *cluster* i

$\vec{x}_i$ Observation *i* du *data-set*

B Nombre d'observations par bloc d'E/S

d Dimension du problème

E Erreur moyenne de l'algorithme

e_i Erreur du *cluster i*

F Sous-ensemble des propriétés à tester pour la division d'un nœud

I Nombre d'itérations nécessaires à l'algorithme *K-means* pour converger

$I/O_{cost\_per\_iter}$ Coût en E/S d'une itération de *K-means*

it_{max} Nombre maximal d'itérations

K Nombre de clusters

M Nombre d'observations qui peuvent être contenues en mémoire

N Nombre d'observations dans un *data-set*

n Nombre d'observations dans un arbre de décision.

R_b Bloc de randomisation

$R_{I/O}$ Réduction du coût en E/S

S Taille réelle de la mémoire disponible

$sepval$ Index de séparation des *clusters*

T Nombre de classifieurs

$T_{<alg>}$ Temps d'exécution de l'algorithme alg

v_i Distance moyenne entre un centre de cluster $\vec{\mu}_i$ et les observations qui lui sont affectés

$V_{I/0}$ Volume des E/S.

Chapitre 1

Introduction

Sommaire

1.1 Contexte

De nos jours, le volume de données mondial produites par l'activité humaine augmente de manière exponentielle. Selon l'*International Data Corporation (IDC)*, 143 zettaoctets seront produits en 2024 [116]. La part des données générées sur les plateformes embarquées et l'Internet des objets ou *Internet of Things* (IoT) représente selon l'IDC la moitié du volume total de données numérisées. Un véhicule autonome produira 40 To de données toutes les 8h de circulation selon le CEO d'Intel [97]. Dans ce contexte, le stockage des données et la mobilisation des ressources mémoire pour les traiter représentent des thématiques de recherche importantes pour une extraction de connaissance efficace. Au vu du gros volume et de la complexité des données collectées, l'extraction de connaissance au niveau des plateformes embarquées nécessite généralement le traitement des données

brutes par des algorithmes d'apprentissage automatique ou MACHINE LEARNING (ML) [123].

L'apprentissage automatique du modèle extrait de la connaissance à partir de données observées (comment classer les données ? quels sont les groupes de données existants ? etc.). Le modèle est ensuite appliqué sur de nouvelles données, qui n'ont jamais été observées, afin de déduire des informations sur ces données : il s'agit de la phase d'inférence. Le ML requiert une puissance de calcul et une capacité mémoire importantes [157]. Par conséquent, ces applications sont généralement exécutées sur des calculateurs puissants [157]. Aujourd'hui, de nombreux fournisseurs de *cloud* offrent des plateformes dédiées au ML, appelées les *Machine Learning as a Service (MLaaS)* [118]. Google a par exemple lancé en 2014 l'API *Prediction*, Microsoft a proposé *Azure Machine Learning*, suivis en 2015 par Amazon avec *AWS Machine Learning* [118]. Bien que ces plateformes présentent de bonnes performances, elles requièrent l'envoi des données collectées des plateformes embarquées vers le *cloud*. Ainsi, le traitement de données sur les MLaaS présentent certaines contraintes :

1. **La confidentialité** : les données collectées sur ces plateformes peuvent être sensibles. C'est notamment le cas lorsqu'il s'agit de données médicales [104], ou personnelles telles que définies par l'Union Européenne dans *The UE General Data Protection Regulation* [147], ou toute autre donnée sensible industrielle ou militaire [42]. Par conséquent, leur envoi vers le *Cloud* pour qu'elles y soient traitées soulève des problèmes de sécurité et de confidentialité [18, 162, 161].
2. **Le coût de communication** : le traitement des données implique leur envoi sur le *cloud* via le réseau. Cela engendre des latences supplémentaires, de la consommation de bande passante et de la consommation énergétique [153]. Selon [18], 70% de la consommation énergétique des dispositifs IoT est due aux communications, ce qui rend cette contrainte particulièrement critique.

Dans ce contexte, plusieurs études s'orientent vers l'apprentissage automatique directement sur les plateformes de collectes [153]. Ce nouveau paradigme, appelé l'EDGE INTELLIGENCE (EI), émerge [65, 131] afin de remédier aux contraintes d'un traitement distant et sera selon [106] un élément vital dans le développement des réseaux 6G.

1.1.1 Paradigme de l'*Edge* Intelligence

L'*Edge Intelligence* (EI) est définie par la Commission Internationale d'Électronique (IEC) comme étant le procédé qui consiste à acquérir des données, les stocker et les traiter avec des algorithmes de ML sur l'*Edge*. Ce dernier désigne les dispositifs « proches » de la

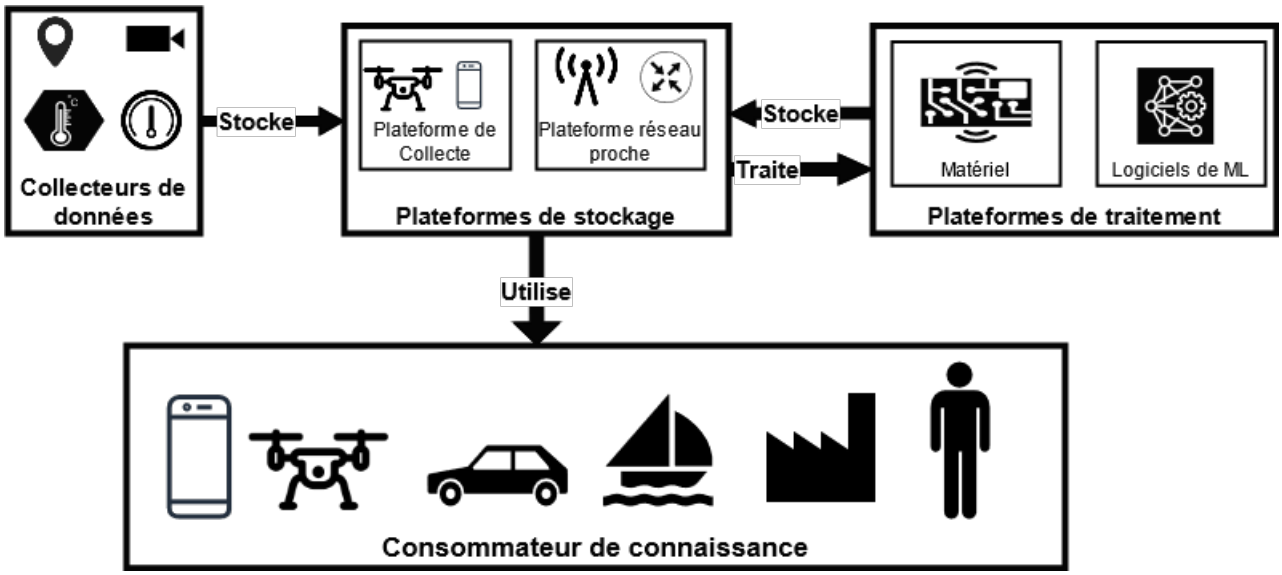


FIGURE 1.1 – *Edge* Intelligence, adaptée de [153]

source de données [32], ce qui comprend les dispositifs de collecte (*smartphones*, caméras, accessoires de domotique, etc.), et les dispositifs réseau qui relient les dispositifs de collecte au *cloud* [32]. La figure 1.1 montre le fonctionnement général de l'EI. Les données sont collectées à l'aide de capteurs (position, température, pression, vidéo, etc.) embarqués sur des dispositifs divers (drone, *smartphone*, voiture, etc.). Les données collectées sont stockées sur le dispositif lui-même ou sur un dispositif réseau proche, où elles seront également traitées pour en extraire de la connaissance. La connaissance est stockée soit sur la plateforme de collecte, soit sur un dispositif réseau proche, puis utilisée par la plateforme où ont été collectées les données ou par un dispositif tiers/humain.

Selon [162], l'EI peut se décomposer en 7 niveaux suivant les dispositifs dans lesquels elle intervient, montrés figure 1.2 :

1. Co-inférence *Edge-Cloud* : l'apprentissage se fait entièrement sur le *cloud* et l'inférence se fait de manière coopérative entre le *cloud* et les dispositifs *Edge*. Cela signifie que les données sont partiellement envoyées vers le *cloud* ;

2. Co-inférence *In-Edge* : l'apprentissage se fait sur le *cloud* et l'inférence se fait sur la plateforme de collecte ou des plateformes « proches » de celle-ci. L'avantage, ici, par rapport au niveau 1, est la baisse de consommation de bande passante, vu que les données d'inférence ne sont pas envoyées jusqu'au *cloud* ;

3. Inférence sur dispositif : l'entraînement du modèle se fait sur le *cloud*, et l'inférence se fait entièrement sur le dispositif. L'avantage est que la consommation de bande passante est nulle une fois que le modèle appris est déployé. Par ailleurs, en terme

de sécurité, les données d'inférence ne sont pas partagées avec d'autres dispositifs ;

4. Co-apprentissage *Cloud-Edge* : dans ce niveau, les deux tâches (apprentissage et inférence) peuvent être effectuées de manière collaborative sur le *cloud* et l'*edge* ;
5. Tout sur *Edge* : les deux tâches s'effectuent sur des plateformes proches des dispositifs de collecte ;
6. Co-apprentissage *Edge*-Dispositif : la tâche d'apprentissage se fait de manière collaborative entre l'*Edge* et le dispositif de collecte ;
7. Tout sur dispositif : l'apprentissage se fait entièrement sur le dispositif. Dans ce niveau, les données ne sont pas partagées, ce qui est avantageux. De plus, la consommation de bande passante est nulle, et ce pour l'apprentissage et l'inférence.

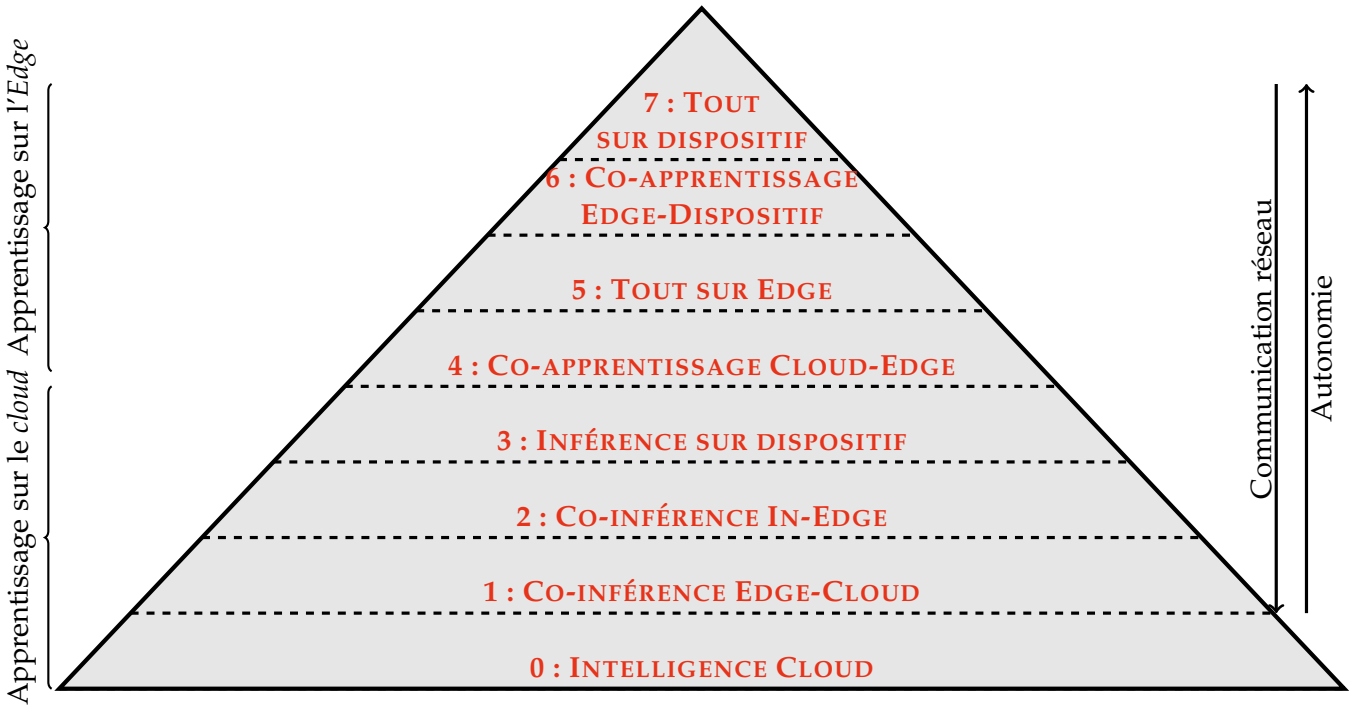


FIGURE 1.2 – Niveaux de l'EI, adaptée de [106] et [162]

La figure 1.2 montre que plus le traitement de données est proche du dispositif de collecte, plus ce dernier acquiert de l'autonomie. Par ailleurs, cette acquisition d'autonomie par les dispositifs de collecte permet de réduire leurs consommation de bande passante. Dans le cadre de cette thèse, nous nous intéressons au cas de l'apprentissage sur les dispositifs embarqués (niveau 7).

1.1.2 Problématiques soulevées par l'EI

Au regard des contraintes matérielles des plateformes de collecte et des dispositifs *edge*, la conception de modèles de ML respectueux des ressources disponibles sur ces plateformes est l'un des défis majeurs de l'EI [162]. Ci-après, nous expliquons ces contraintes matérielles.

Contraintes énergétiques :

Les systèmes embarqués présentent des contraintes énergétiques notables. En effet, la plupart de ces dispositifs sont alimentés par batteries, c'est le cas par exemple pour les *smartphones*, les drones, etc. [120]. Au vu de la lenteur de la tâche d'apprentissage comparée à la durée d'autonomie des batteries sur les systèmes embarqués, la mise en œuvre des algorithmes de ML directement sur ces plateformes aura pour effet la baisse de leurs temps de disponibilité [161]. Le second impact est la surchauffe de ces dispositifs causée par l'intensité des calculs effectués lors de l'apprentissage [77, 161], ce qui peut causer l'usure rapide des composants électroniques ou des départs de feux sur le matériel. L'un des mécanismes souvent employés pour limiter cette surchauffe est la réduction de la fréquence de l'horloge du processeur [161]. Néanmoins, cela a pour effet de ralentir considérablement le temps d'apprentissage.

Contraintes mémoire :

L'espace mémoire sur les systèmes embarqués est relativement faible par rapport au volume de données qui sont amenées à y être traitées [16], notamment avec l'avènement de l'EI [92]. En effet, parmi les applications couramment utilisées dans l'EI, on trouve le traitement d'images. Dans [161], les auteurs ont évalué l'empreinte mémoire de plusieurs modèles, à base de réseaux de neurones, et montrent que celle-ci varie entre 5 et 548 Mo. Afin de mettre en perspective cette empreinte mémoire par rapport au matériel disponible sur le marché, nous montrons dans le tableau 1.1 les volumes mémoire et stockage disponibles sur quelques plateformes embarquées récentes. On remarque que les volumes mémoires disponibles sur ces plateformes varient entre 0.5 et 8 Go. Le volume mémoire qui peut y être embarqué peut difficilement être augmenté pour des raisons technologiques [**mutlu2013**]. La technologie utilisée pour la mémoire de travail sur ces plateformes est la *Dynamic Random Access Memory* (DRAM). Celle-ci est limitée en terme de densité (nombre de cellules mémoire par unité de surface), car l'augmentation de la densité est obtenue grâce à la réduction de la taille des transistors. Or, en-deçà d'une certaine taille de transistor, les courants de fuite augmentent et rendent le rafraîchissement des données

Carte embarquée	Exemples d'applications	Volume mémoire (Go)	Volume de stockage
Raspberry Pi 4	Surveillance de la qualité de l'air [46], agriculture [53]	2, 4 ou 8	1 Slot microSD
Raspberry Pi Zero	Reconnaissance faciale [96], contrôle de taux de glycémie [3]	0.5	1 Slot MicroSD
Nvidia Jetson	Localisation d'objets [126], détection de piétons [22], atterrissage autonome d'un drone [156]	2, 4 ou 8	16 ou 32 Go
Qualcomm Robotics RB5 Kit	Détection de remplissage d'un parking [136]	8	1 To Nand flash

TABLE 1.1 – Caractéristiques mémoire et stockage des plateformes embarquées récentes [84]

plus fréquent. Par ailleurs, plus le volume de la DRAM est élevé, plus l'énergie nécessaire pour un rafraîchissement des données est importante. Par conséquent, dans le contexte d'un dispositif contraint en terme d'énergie, l'augmentation du volume mémoire est difficilement réalisable. En somme, le tableau 1.1 montre que le volume de mémoire disponible sur les plateformes embarquées est limité en comparaison aux plateformes MLaaS et au regard de l'empreinte mémoire des modèles d'apprentissage.

1.1.3 Le projet Systèmes Autonomes en Milieu Marin (SAMM)

Depuis l'année 2010, le domaine de la marétique fait son apparition pour désigner l'ensemble des systèmes informatiques et électroniques utilisés dans la gestion et l'automatisation des opérations relatives aux activités maritimes, fluviales et portuaires [31]. Ce domaine est déjà une réalité dans la région Bretagne où il s'appuie sur les compétences numériques historiques et la volonté de la région d'en faire un pilier de son développement durable [31]. Le Conseil Économique, Social et Environnemental Régional a pour objectif d'explorer la place de la marétique dans 7 cas d'usages :

- Connaître, surveiller et protéger les écosystèmes marins, en tenant compte de leurs interactions avec les activités humaines ;
- Concevoir et produire un navire, assurer sa maintenance ;
- Naviguer, être marin ;
- Exploiter et consommer les ressources marines alimentaires ;
- Renforcer et gérer les flux de personnes et de biens à l'interface entre la terre et la mer ;
- Produire et distribuer des énergies marines renouvelables ;
- Pratiquer la mer, expliquer et transmettre la culture maritime.

Le programme Systèmes Autonomes en Milieu Marin (SAMM) est l'un des programmes

FIGURE 1.3 – Drones marins

soutenus pas la région Bretagne à ce titre. L'objectif de ce programme est de développer une filière d'excellence dans le domaine des Sciences et Technologies de l'Information et de la Communication (STIC). Il consiste en un soutien régional dans 5 domaines : intelligence, capteurs et algorithmes, systèmes embarqués, *data mining* et enfin, interactions homme/machines et communications. Le programme SAMM met à disposition des projets soutenus, entre autres, des infrastructures de test (bases de test marines, volières) et des platefomes et équipements (drones roulants, flottants, sous-marins, volants) [36]. Cette thèse est financée à hauteur de 50% par ce programme, le reste étant financé par un Contrat Doctoral d'Établissement (CDE).

1.2 Problématique : réduction du volume des E/S dans l'EI

L'EI est aujourd'hui un paradigme largement envisagé et investi, et pourrait s'imposer dans les années à venir comme modèle de production de connaissances sur les équipements IoT, censés acquérir de plus en plus d'autonomie. L'un des défis de l'EI est de réaliser l'entraînement directement sur les plateformes en tenant compte des limitations en terme de ressources [162]. Dans le cadre de cette thèse, nous nous intéressons à la gestion des accès à la hiérarchie mémoire lors de la phase d'apprentissage des algorithmes de ML. En effet, l'augmentation constante du volume de données d'apprentissage, combinée aux fortes contraintes en terme d'espace de travail sur les systèmes embarqués, rend la tâche complexe.

Outre les problèmes liés à l'architecture des plateformes embarquées, les algorithmes classiques de ML sont conçus avec comme hypothèse que les données d'apprentissage

peuvent être entièrement contenues en mémoire [110]. Or, dans le cas où l'espace mémoire disponible est insuffisant, la gestion des accès aux données incombe au système d'exploitation. Classiquement, la gestion de tels cas de figure par le système d'exploitation se fait par les mécanismes de mémoire virtuelle et de *swap*, qui consiste à étendre l'espace d'adressage physique en utilisant l'espace de stockage secondaire [17]. Cependant, le temps d'accès au stockage secondaire est environs 2500 fois plus lent que l'accès à la DRAM [15]. Au vu des différences entre les algorithmes d'apprentissage, notamment en terme de motifs de parcours de données, il est difficile de proposer une méthode générique qui réduit les mouvements entre la mémoire principale et l'espace *swap*. Par conséquent, nous nous sommes restreint dans cette thèse à deux cas d'études : le *K*-MEANS, souvent utilisé pour la segmentation d'images, la détection d'intrusions, la compression de données, etc. [1], et les RANDOM FORESTS, massivement utilisés pour la détection d'intrusions dans les systèmes informatiques [117], mais aussi pour la tâche de classification dans divers domaines : les sciences de l'environnement, la génétique, la bionformatique, etc. [142]. Le choix des cas d'étude s'est fait sur la base de critères liés au contexte de l'EI :

— La popularité : un algorithme déjà répandu pour accomplir des tâches d'apprentissage est plus susceptible d'être testé et intégré dans l'EI ;
— L'adéquation pour un *data-set* volumineux : dans le contexte de l'augmentation du volume des *data-set*, il est plus judicieux d'utiliser des algorithmes capables de répondre à ce besoin ;
— La rapidité : étant données les contraintes temporelles qui peuvent exister sur les systèmes embarqués, il est important que le temps d'exécution soit le plus réduit possible.

Comme il sera détaillé dans cette thèse, *K*-MEANS et RANDOM FORESTS itèrent plusieurs fois sur l'ensemble des données d'apprentissage. Dans le cas d'un environnement contraint en mémoire, mais traitant d'importants volumes de données, cela cause de multiples mouvements de données entre la mémoire physique et l'espace de *swap*. Or, au vu du coût des accès à la mémoire secondaire, une large proportion du temps d'apprentissage des algorithmes de ML est consacrée à ces E/S. Nous formulons alors la problématique de cette thèse comme suit : **comment réduire le volume des E/S des algorithmes de ML, notamment pour les cas d'étude que nous avons choisis, pour permettre leur intégration dans les plateformes embarquées ?**

1.3 Contributions

Pour chacun des cas d'étude choisis, la démarche générale suivie pour répondre à la problématique est la suivante :

1. Analyse et caractérisation théorique et expérimentale des motifs d'E/S de l'algorithme, pour identifier les parties les plus intensives en E/S ;
2. Identification, au niveau algorithmique, des parties intensives en E/S dans le contexte d'un environnement restreint en mémoire ;
3. Proposition d'une solution algorithmique pouvant réduire le volume des E/S.

1.3.1 Contribution 1 : réduction des E/S pour l'algorithme du *K-means*

Le premier algorithme auquel nous nous sommes intéressée est l'algorithme de *clustering K-means*. En effet, l'analyse de l'algorithme *K-means* du point de vue accès à l'espace *swap* nous a permis de quantifier le coût des E/S induit par le chargement complet du *data-set* en mémoire lorsque l'espace mémoire disponible est de plus en plus petit. Le problème que nous avons identifié sur cet algorithme est **le fait de parcourir plusieurs fois le *data-set*** pour redistribuer ses données sur les *clusters* (l'objectif de la redistribution est l'amélioration de la qualité du *clustering*). Ces multiples parcours augmentent considérablement le volume de données *swappées*. À titre d'exemple, pour un *data-set* 4 fois plus volumineux que l'espace mémoire disponible, le pourcentage de temps passé à faire des E/S est de 70%. Notre objectif est de réduire le nombre de parcours du *data-set*, ce qui aura pour conséquence de réduire le temps où l'algorithme est en attente de données devant être chargées depuis le stockage secondaire.

L'idée dans cette contribution est d'appliquer l'algorithme *K-means* sur des portions de données qui peuvent être contenues en mémoire (*chunk*), puis de combiner les résultats obtenus pour chaque portion pour aboutir à un *clustering* de l'ensemble du *data-set*. Ce faisant, le transfert des données d'apprentissage vers la mémoire principale ne se fait qu'une seule fois. L'évaluation de cette méthode a montré que le pourcentage de temps d'E/S ne dépasse pas les 4% et réduit le temps d'apprentissage de 60% en moyenne pour l'ensemble des tests réalisés.

Cette contribution a fait l'objet d'une publication dans la conférence IEEE MASCOTS 2019 [133].

1.3.2 Contributions 2 et 3 : réduction des E/S pour l'algorithme des *Random Forests*

L'analyse de l'algorithme de construction d'une forêt aléatoire nous a permis de mettre en évidence deux problèmes à l'origine d'une proportion de temps d'E/S, qui peut atteindre les 80%, lorsque le *data-set* est plus grand que l'espace mémoire disponible :

— Étant donné un bloc du *data-set*, ses éléments sont accédés à différentes étapes de la construction d'un arbre de décision. Par conséquent, cet algorithme exhibe une faible localité spatiale ;

— La construction d'un arbre de décision se fait sur un sous-ensemble (*bootstrap*) du *data-set*. Or, l'hypothèse de l'algorithme classique est que la totalité du *data-set* est chargée en mémoire avant le début de construction des arbres de décision, ce qui fait qu'une proportion importante des données est parcourue inutilement et augmente ainsi le volume d'E/S.

Pour répondre à ces problèmes, nous avons conçu 2 solutions complémentaires qui réduisent le volume d'E/S à la construction d'une RF.

Contributions 2 : réorganisation du *data-set*

Pour améliorer la localité spatiale lors de la construction de la forêt aléatoire, notre première contribution consiste à réorganiser le *data-set* sur le périphérique de stockage de manière à ce que les données susceptibles d'être accédées dans la même fenêtre temporelle lors de la construction d'un arbre soient stockées dans des blocs voisins. Cette idée est motivée par une propriété des arbres de décision d'une même forêt, observée expérimentalement sur certains ensembles de données. En effet, si une paire d'éléments du *data-set* est accédée dans une même fenêtre temporelle pendant la construction d'un arbre de décision, ses éléments ont une forte probabilité d'être accédés dans la même fenêtre temporelle pour la construction des autres arbres de décision. Sur la base de cette propriété, nous avons utilisé le premier arbre de la forêt aléatoire pour déduire des éléments qui seront probablement accédés dans un intervalle de temps court. Ensuite, nous avons réorganisé le *data-set* de manière à regrouper ces éléments sur les mêmes blocs. Une fois le nouveau *data-set* écrit, il est utilisé pour construire les arbres de décision restants.

L'évaluation de cette méthode a montré qu'elle permet de réduire le temps de construction d'une forêt aléatoire par rapport à méthode classique de près de 70% en moyenne.

Une version préliminaire de cette méthode a été publiée à la Conférence francophone d'informatique en Parallélisme, Architecture et Système (COMPAS). Un article portant

sur cette contribution et la suivante a également été soumis à la revue *IEEE Transactions on Computers*.

Contribution 3 : accès à la demande aux données du *data-set*

Pour répondre au problème de mouvements (*swap*) des données inutiles entre la mémoire principale et le stockage secondaire pour la construction de la *Random Forest*, nous nous sommes basée sur deux idées :

1. Charger les éléments nécessaires au traitement de chaque nœud, et non pas tout le *data-set* au début de la construction : de cette manière, nous évitons le volume *swappé* inutilement qui résulte du chargement des données qui ne font pas partie des données d'un arbre de décision ;
2. Dans le cas où le volume de données chargées reste supérieur à l'espace mémoire disponible, nous effectuons le traitement des données de chaque nœud par parties, et nous combinons les résultats obtenus pour chaque partie, ce qui permet d'éviter le parcours multiple de certaines données.

L'évaluation de cette méthode a montré qu'elle permet de réduire en moyenne le temps de construction d'un arbre de décision de 60%.

Cette contribution a fait l'objet d'une publication dans la conférence ACM Symposium on Applied Computing 2021 [134].

1.4 Plan de la thèse

Le présent document est organisé comme suit :

— Le chapitre 2 définit les concepts généraux dont nous aurons besoin tout au long de ce manuscrit, et présente une synthèse bibliographique liée à la réduction des E/S dans l'EI ;

— Le chapitre 3 décrit la contribution 1. Nous prenons l'algorithme de *clustering K-means* comme premier cas d'étude. Nous caractérisons son motif d'accès au stockage, et analysons son algorithme pour déterminer la partie responsable de ce motif. Nous proposons une solution pour y répondre et terminons par une évaluation de cette dernière ;

— Le chapitre 4 porte sur les contributions 2 et 3, qui concernent l'optimisation des RF. Nous avons identifié deux problèmes qui causent un volume conséquent d'E/S. Ce chapitre est donc structuré pour répondre à chacun des problèmes ;

— Le chapitre 5 clôt cette thèse et propose quelques perspectives de recherche qui découlent de ce travail.

Chapitre 2

L'*Edge* Intelligence face à la contrainte mémoire

Introduction

Dans ce chapitre, nous commençons par donner une vue globale du domaine de l'*Edge* Intelligence et de ses champs d'application.

« L'Intelligence » des dispositifs *Edge* émanant des algorithmes de *Machine Learning*, nous définissons et passons en revue la taxonomie de ces algorithmes, en mettant l'accent sur les cas d'étude choisis pour cette thèse, à savoir le *K-means* et les *Random Forests*. L'Edge Intelligence est un domaine en plein émergence qui soulève plusieurs défis scientifiques. Nous décrivons dans la section 2.3 ces principaux défis. Nous nous focalisons par la suite sur la problématique de la contrainte mémoire en commençant par décrire la hiérarchie mémoire dans le paysage de l'*Edge* Intelligence, puis synthétisons les travaux visant à répondre à la problématique de la contrainte mémoire.

Sommaire

2.1 L'*Edge* Intelligence

2.1.1 Introduction à l'Edge Intelligence

Bien que le terme *Edge Intelligence* n'est apparu que récemment, les prémisses de ce paradigme de traitement de données remontent à 2009, avec un prototype de reconnaissance de commande vocale directement sur les appareils mobiles lancé par Microsoft [162]. Dès lors, ce domaine de recherche n'a cessé de prendre de l'ampleur et a été mentionné pour la première fois en 2018 dans la *Gartner Hype Cycle* comme étant à sa phase d'innovation et devrait atteindre son plateau de productivité entre 2024 et 2029 [102].

À notre connaissance et selon [162], il n'existe pas de définition formelle de l'EI. Néanmoins, la définition la plus répandue dans les milieux académiques et industriels peut être formulée comme suit :

Définition 1. *L'**Edge Intelligence** (EI) se réfère au paradigme dans lequel les algorithmes de Machine Learning s'exécutent localement sur les dispositifs où les données ont été collectées ou des dispositifs à la périphérie des points de collecte [162], dans le but d'améliorer la qualité et la rapidité du traitement de données et de protéger les données [153].*

Objectifs

L'EI est à l'intersection entre l'*Edge Computing* et le *Machine Learning* [162]. L'*Edge Computing* consiste à exploiter la puissance de calcul de plusieurs dispositifs capables de col-

laborer pour traiter les données générées à proximité. Le *Machine Learning*, quant à lui, a pour objectif de « simuler » une intelligence humaine sur des machines en apprenant à partir de données générées [162]. La combinaison de ces deux domaines a, alors, plusieurs objectifs :

- Assurer la proximité entre la tâche de prise de décision et les dispositifs de collecte des données [18] : traditionnellement, les tâches d'apprentissage et de prise de décision sur les dispositifs embarqués sont accomplies sur le *cloud*. Or, le bon déroulement de ces tâches nécessitent la disponibilité d'une connexion réseau pour à la fois, le transfert des données brutes vers le *cloud* pour l'apprentissage et la récupération des résultats (modèle appris ou application du modèle sur une donnée) depuis le *cloud*. Cette dépendance au réseau internet est problématique lorsque le réseau est indisponible, instable ou congestionné. Par conséquent, l'émergence de l'EI vise, entre autres, à s'affranchir de cette dépendance.
- Assurer la sécurité et la confidentialité des données collectées : en effet, il est montré dans [137] et dans [138] que le *cloud* présente de nombreuses vulnérabilités en terme de sécurité et protection des données qui peuvent être regroupées ainsi : *(1)* la technologie de virtualisation et le modèle multi-tenant du *cloud* peut causer la perte de données, *(2)* la confiance entre l'utilisateur (*Edge*) et le fournisseur de *cloud* est limitée, *(3)* les données sont fortement centralisées ce qui accroît le risque de perte de données.
- Réduire la consommation énergétique due aux communications : selon [18], 70% de la consommation énergétiques des dispositifs embarqués, dans le contexte de l'IoT par exemple, est causée par la communication réseau sans-fil, alors que l'autonomie des batteries est limitée. Dans de tels systèmes, la réduction des communications réseaux est importante puisque plus coûteuses que le traitement de données [18].
- Respecter les délais temps réel pour certaines applications : en effet, selon [18], les systèmes intelligents basés sur le *cloud* n'assurent pas le respect des délais d'exécution des applications temps-réel car le temps de réponse dépend de plusieurs paramètres difficiles à prédire.

2.1.2 Architecture de l'EI

Dans cette partie, nous nous basons sur [162] et [153] pour décrire les architectures de l'EI. La figure 2.1 montre une comparaison entre l'architecture de l'intelligence basée sur le *cloud* et l'architecture de l'EI avec ses variantes. Avant de décrire ces architectures, nous commençons par classer les dispositifs présents sur la figure afin de mieux com-

prendre les architectures qui seront présentées par la suite. Nous avons identifié, à partir des références bibliographiques [18, 77, 153, 162], trois catégories d'entités dans ces architectures qui sont les suivantes :

— **Le *Cloud* :** c'est une infrastructure qui déploie des ressources de calcul et de stockage importantes, configurables et partagées entre plusieurs utilisateurs.
— **Le *Fog* [93] :** il désigne l'ensemble des équipements du réseau reliant les dispositifs de collecte de données au *cloud* et qui sont capables de faire du traitement et du stockage de données. Nous pouvons citer comme exemples d'équipements *Fog* : les *switchs*, les routeurs, les passerelles et les stations de base.
— **Le dispositifs de collecte, objets connectés (*End point Devices*) :** ce terme désigne l'ensemble des dispositifs embarqués qui ont pour rôle de collecter les données à partir de leur environnement grâce à des capteurs et des actionneurs [18]. Nous pouvons citer dans cette catégorie les *smartphones*, les caméras de surveillance, les drones, les divers accessoires de domotique, les véhicules, etc.

La figure 2.1 dresse un comparatif entre une architecture où l'intelligence est basée exclusivement sur le *cloud* et une architecture où l'intelligence est déployée sur la partie *Edge*. Dans le premier cas de figure, la donnée est acheminée depuis les points de collecte vers le *cloud* où le modèle « intelligent » sera construit et interrogé pour en extraire de l'information. Dans le deuxième cas en revanche, il existe trois variantes d'architecture EI :

— L'architecture centralisée, cas (1) sur la figure : dans ce cas de figure, un sous-ensemble des équipements de l'*Edge* considère l'un des équipements (appartenant au *Fog* ou aux dispositifs de collecte) comme élément central. Ces équipements lui envoient alors leurs données pour que le modèle « intelligent » y soit formé, stocké et interrogé. Dans la figure 2.1, nous imaginons un scénario en domotique, où plusieurs équipements à usage quotidien sont liés au *smartphone* de l'utilisateur qui joue le rôle d'élément central. Nous pouvons alors imaginer qu'à partir des informations collectées par ces équipements, les habitudes quotidiennes de l'utilisateur peuvent être apprises pour déduire un système de contrôle de sa consommation énergétique.
— L'architecture décentralisée, (3) sur la figure : dans ce cas, le modèle intelligent est propre à chaque équipement de collecte. L'intelligence y est formée à partir des données qu'il collecte et est destinée à son usage propre.
— L'architecture hybride, cas (2) sur la figure : dans ce type d'architectures, les équipements de collecte et les équipements *Edge* collaborent pour développer un modèle intelligent. Cela peut se justifier par une dépendance entre les données des équipements de collecte pour aboutir au modèle intelligent, ou encore par l'insuffi-

sance des ressources sur les équipements de collecte, d'où le besoin de déléguer cela à un équipement *Fog*. Dans la figure 2.1, le scénario montré est celui de la gestion du trafic en ville. Nous imaginons plusieurs équipements de collecte tels que les véhicules, les caméras de surveillance, les feux tricolores qui collaborent avec des équipements du *Fog* afin de développer une intelligence déployée sur l'ensemble des équipements pour décongestionner le trafic routier.

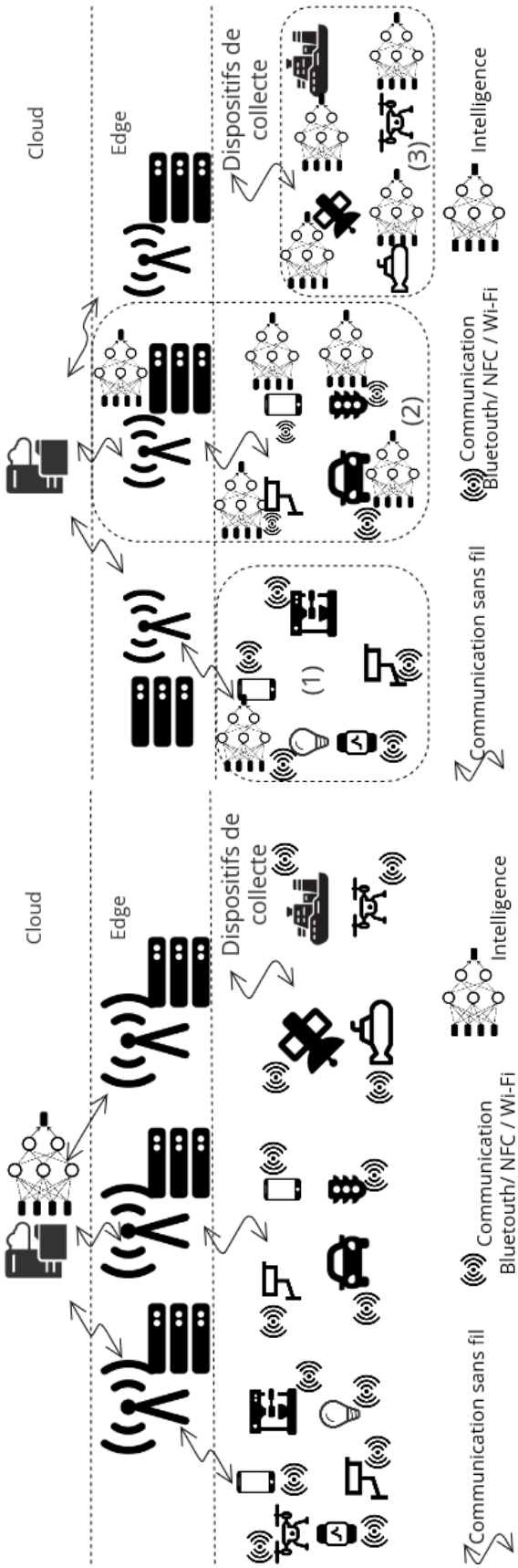


FIGURE 2.1 – Comparaison des architectures intelligence basée sur *cloud* et *edge* Intelligence, adaptée de [162] et [153]

2.1.3 Taxonomie de la littérature de l'EI

Selon [153], les trois composantes principales d'une application intelligente sont les données, l'algorithme et les calculs. Ces trois composantes se traduisent dans un environnement *Edge* par les champs de recherches suivants :

— L'*Edge Caching :* cela consiste à définir une stratégie pour la collecte et le stockage des données générées sur les dispositifs *Edge* ou reçues depuis les points de collecte. Concrètement, cela revient à répondre aux trois problématiques suivantes : parmi les données collectées, lesquelles doivent être stockées ? Où stocker les données étant données une infrastructure d'EI ? Comment stocker les données ?

— L'entraînement sur l'*Edge* : cela consiste à définir une stratégie qui permet d'extraire de la connaissance depuis les données collectées. Les travaux de recherche qui s'intéressent à l'entraînement sur l'*Edge* visent à répondre aux problématiques suivantes : quelle est l'architecture à utiliser pour l'entraînement ? Comment accélérer l'entraînement ? Comment optimiser la procédure d'entraînement selon un objectif donné (réduction de la consommation énergétique et/ou mémoire, sécurisation des données, etc) ? Comment évaluer la qualité du modèle obtenu ?

— L'inférence sur l'*Edge* : cela consiste à utiliser le modèle appris afin de tirer effectivement de la connaissance sur une nouvelle observation. Les problématiques auxquelles répondent les travaux qui s'inscrivent dans cette composante de l'EI sont : comment faire en sorte qu'un modèle appris soit applicable sur un dispositif (ce dispositif peut être le même que celui où le modèle a été appris, ou sur un dispositif tiers) ? Comment accélérer la tâche d'inférence ?

— L'*offloading* sur *Edge* : cette branche de l'EI intervient lorsque l'infrastructure met en œuvre le calcul distribué, ce qui sous-entend que si un dispositif ne dispose pas de suffisamment de ressources pour accomplir une tâche de l'EI, celle-ci peut être déléguée à un autre dispositif. Les travaux de cette branche de l'EI visent, donc, à définir une stratégie de distribution des tâches sur les dispositifs de l'infrastructure.

2.1.4 Positionnement architectural et applicatif de la thèse dans le paysage de l'EI

À travers la revue bibliographique des architectures de l'EI, ainsi que des 4 principaux champs de recherche de l'EI, nous restreignons le contexte de cette thèse à ce qui suit :

— D'un point de vue architectural : nous nous restreignons au modèle décentralisé en posant l'hypothèse suivante : l'application visée ne nécessite pas de données collectées sur d'autres équipements ;

— D'un point de vue applicatif : nous nous intéressons à l'entraînement sur *Edge*. L'hypothèse que les données collectées sur l'équipement suffisent à développer un modèle intelligent nous motive à nous diriger vers une solution où le modèle est appris directement sur le dispositif, ce qui permet de nous affranchir de la gestion des communications et de la sécurité.

2.2 Concepts généraux liés au Machine Learning

2.2.1 Introduction au machine learning

Définition 2. ***Machine Learning ML*** *ou* ***Apprentissage automatique*** *en français se réfère à la capacité d'un programme d'apprendre, c'est-à-dire, modifier son comportement sur la base d'une expérience, sans que cette modification ne soit explicitement programmée [90].*

Autrement formulé, un algorithme d'apprentissage a pour but d'apprendre à travers l'expérience E à accomplir les tâches T d'une manière performante. À titre d'exemple, dans le cas d'un problème de reconnaissance d'écriture manuscrite, la tâche à accomplir est la classification de mots dans des images, la mesure de performance est le pourcentage de mots correctement classifiés, et l'expérience est l'ensemble des mots manuscrits (*data-set*, voir la définition 3 page 36) avec leurs classifications.

Définition 3. ***Data-set*** *ou* ***Jeu de données*** *en français : collection de valeurs quantitative et/ou qualitatives. Chaque valeur est associée à une variable et une observation. Ainsi, un data-set peut être vu comme étant une matrice où les lignes sont des observations et les colonnes des variables qui caractérisent ces observations [148].*

Classification des algorithmes de ML

Dans la littérature, les algorithmes de ML sont classés selon les caractéristiques des données d'entrée (*data-set*) sur lesquels ils agissent. En somme, cette classification dépend de la présence - ou non - d'étiquettes dans le *data-set* qui représentent l'information que l'on veut prédire. Les catégories d'algorithmes de ML sont les suivantes :

L'apprentissage supervisé : dans l'apprentissage supervisé, les observations du *data-set* sont étiquetées par les valeurs que l'on cherche à prédire par l'algorithme. Formellement, il peut être défini comme suit.

Définition 4. *Soient N observations $\{X\}_{1,\dots,N}$ et leurs étiquettes $\{Y\}_{1,\dots,N}$ (l'étiquette représente l'information que l'on souhaite prédire). On suppose qu'il existe une fonction fixe et inconnue*

$\phi : X \to Y$. L'objectif de l'apprentissage est alors d'utiliser les observations du data-set afin de déterminer une fonction $f : X \to Y$ telle que pour tout couple $(X, \phi(X))$, $f(X) \approx \phi(X)$ [7].

Les problèmes auxquels répond l'apprentissage supervisé. Il est employé pour répondre aux problèmes suivants :

1. La classification : on parle de problème de classification lorsque les étiquettes à prédire sont discrètes. Lorsque le nombre de classes possibles est de 2, on parle de classification binaire. Lorsqu'il est supérieur à 2, alors il s'agit d'une classification multiclasses.
2. La régression : lorsque les étiquettes à prédire ne sont pas entières, il s'agit d'un problème de régression.

L'apprentissage non supervisé : contrairement à l'apprentissage supervisé, dans l'apprentissage non supervisé, les observations ne sont pas étiquetées. Par conséquent, les algorithmes doivent extraire de manière autonome les relations qui existent entre les données. La formalisation d'un problème d'apprentissage non supervisé est donnée comme suit [7].

Définition 5. *Soit N observations $\{X\}_{1,\dots,N}$. Il s'agit d'apprendre une fonction f qui vérifie certaines propriétés sur ces observations.*

Comme on peut l'observer, cette définition est vague comparée à celle de l'apprentissage supervisé. En effet, dans l'apprentissage non supervisé, on ne connaît pas a priori la structure du *data-set* et par conséquent les informations que l'on va en extraire.

Les problèmes auxquels répond l'apprentissage non supervisé. Il est utilisé pour résoudre les problèmes suivants :

1. Le *Clustering* ou partitionnement [150] : l'objectif est de regrouper les observations dans des partitions (*clusters*) disjointes sur la base de la similarité, de telle sorte que les observations qui sont dans le même *cluster* soient similaires entre elles et soient non-similaires avec les observations des autres *clusters*. Autrement dit, l'objectif est de découvrir la structure ou les motifs existants dans un ensemble d'observations. Par exemple, dans [89], les auteurs ont pour objectif de prédire la trajectoire de débris marins et d'algues afin d'améliorer le processus d'élimination des déchets océaniques. Pour ce faire, les auteurs suivent les trajectoires d'objets crées pour mimer le comportement des débris marins. Un modèle est entraîné afin de prédire

la trajectoire d'un objet selon ses caractéristiques physiques en utilisant le *clustering* sur les trajectoires suivies.

2. La réduction de dimension [150] : pour un *data-set* donné, l'objectif est de réduire le nombre de variables (dimension) des observations en passant du nombre de variables original vers un plus petit nombre de variables orthogonales les une aux autres et qui soient représentatives du *data-set* d'origine. À titre d'exemple, dans [140], les auteurs ont pour objectif d'exploiter une base de données qui contient des paramètres physiques, chimiques et micro-biologiques (15 paramètres au total) sur 30 ans, des eaux de surface et souterraines dans la région Provence-Alpes-Côte d'Azur (France), pour mesurer leurs qualités pour la consommation humaine. Or, l'évolution dans le temps de cette base de données en terme d'ajout de nouveaux paramètres mesurés l'a rendue difficilement exploitable. Par conséquent, les auteurs ont utilisé la réduction de dimensions sur un sous-ensemble de cette base de données, et ont identifié 6 composantes principales représentatives des données. Chacune des composantes regroupe un sous-ensembles des paramètres de départ.

L'apprentissage semi-supervisé : il est à mi-chemin entre l'apprentissage supervisé et l'apprentissage non supervisé. Le *data-set* sur lequel il agit contient des observations non étiquetées (comme dans l'apprentissage non supervisé) et des observations étiquetées (cas de l'apprentissage supervisé) [26]. Autrement dit, le *data-set* $\{X\}_{1,\dots,N}$ est constitué de deux parties : $\{X_l\}_{1,\dots,l}$ qui sont les observations étiquetées pour lesquels les étiquettes sont $\{Y\}_{1,\dots,l}$, et $\{X_u\}_{l+1,\dots,l+u}$ qui sont des observations non étiquetées. Cette approche d'apprentissage est motivée par deux raisons : (1) le coût et la lenteur de l'opération d'étiquetage des observations qui nécessite l'intervention d'un humain. *Amazon* a, par exemple, lancé en 2005 la plateforme *Amazon Mechanical Turk* basée sur le principe du *Crowd Sourcing* où des tâches d'étiquetages de données sont proposées aux utilisateurs contre rémunération [33] ; (2) sous certaines conditions, la prise en compte des observations non étiquetées améliore la précision du modèle appris [26].

L'apprentissage par renforcement : il intervient dans le contexte d'environnement relié à un agent. Ce dernier reçoit en entrée l'état courant de son environnement et prend une décision sur l'action à exécuter en conséquence. Selon la pertinence de la décision prise, l'agent reçoit une récompense ou une pénalité. Le système tend alors à être de plus en plus performant [7].

NB : l'apprentissage semi-supervisé et l'apprentissage par renforcement ne sont pas

dans le champ d'étude de cette thèse.

2.2.2 Les algorithmes de *clustering*

Dans cette partie, nous décrivons les algorithmes de base de *clustering*, nous mettons en évidence leurs similarités et différences. Selon [154], il existe 26 algorithmes classiques de *clustering* qui peuvent être classés en 9 catégories que nous résumons ci-après :

1. **Le *clustering* basé sur les partitions :** l'idée de base pour cette catégorie est de considérer que chaque *cluster* est défini par un point autour duquel sont centrés les observations affectées à ce *cluster*. Les algorithmes les plus communs pour cette catégorie sont le *K*-means [86] et le *K*-medoids [103]. La différence entre ces deux algorithmes sera mise en évidence dans la description de l'algorithme *K*-means (section 2.2.3). D'autres algorithmes existent dans cette catégorie PAM [72], CLARA et CLARANS [98].

2. **Le *clustering* hiérarchique :** ces algorithmes supposent au départ que chaque observation représente un *cluster*. Par la suite, chaque *cluster* est regroupé avec le *cluster* qui lui est le plus proche selon la définition de distance fixée (voir la définition 7 page 42). Le processus est répété jusqu'à obtention du nombre de *clusters* souhaité. Il existe plusieurs algorithmes classiques de *clustering* hiérarchique : Birch [160], CURE [58], ROCK [113], Chameleon [71].

3. **Le *clustering* basé sur la théorie du flou :** dans les algorithmes de cette catégorie, les observations du *data-set* ne sont pas assignées à des *clusters précis*, autrement dit, l'appartenance d'une observation à un *cluster* n'est pas une variable discrète $\{0, 1\}$, mais une variable continue $[0, 1]$. Dans [155], l'auteur donne plusieurs exemples de fonctions qui permettent d'affecter les observations à des *clusters* selon la théorie du flou.

4. **Le *clustering* basé sur la distribution :** l'idée de cette catégorie d'algorithme est que les observations qui appartiennent à un même *cluster* sont générées selon la même distribution. Cela consiste à supposer que les observations de chaque *cluster* ont été générées par une distribution Gaussienne. L'un des algorithmes les plus communs dans cette catégorie est le GMM proposé dans [115].

5. **Le *clustering* basé sur la densité :** dans cette catégorie, les observations qui appartiennent à une même région de l'espace dont la densité est considérée comme élevée sont supposées appartenir au même *cluster*. L'un des algorithmes les plus répandus dans cette catégorie est le DBSCAN [45].

6. **Le *clustering* basé sur la théorie des graphes :** dans cette catégorie, le problème est représenté comme un graphe où les nœuds sont les observations et les arêtes sont les distances entre les observations. L'algorithme CLICK [129] en est un exemple.

7. **Le *clustering* basé sur les grilles :** dans cette catégorie, l'espace des observations est structuré en grilles tel que chaque cellule de la grille regroupe un sous-ensemble d'observations. L'idée sous-jacente consiste à « résumer » le *data-set* en remplaçant chaque sous-ensemble d'observations d'une cellule par le centre de la cellule. Le *clustering* est effectué en agrégeant les cellules denses adjacentes pour former des *clusters* [2].

8. **Le *clustering* basé sur la théorie fractale :** la définition géométrique d'une structure fractale est une structure qui ne varie pas par changement d'échelle. Les algorithmes basés sur la théorie fractale consistent à supposer des observations de départ comme centres des *clusters* puis d'ajouter de manière incrémentale les autres observations du *data-set* tel qu'une observation est ajoutée à un *cluster* si elle ne fait pas varier la dimension fractale [1] de ce *cluster* [8].

9. **Le *clustering* basé sur les modèles :** dans cette catégorie, la structure géométrique et la distribution de données est imposée au préalable à l'opération de *clustering*. Il s'agit alors de trouver le paramétrage du modèle imposé de manière à ce qu'il corresponde aux observations du *data-set*. L'algorithme COBEWEB [50] est basé sur cette technique.

Le tableau 2.1 dresse un comparatif des algorithmes traditionnels de *clustering* selon les critères de complexité temporelle, et de leur adaptabilité à un gros volume de données [154].

Justification du choix de l'algorithme *K-means*

L'algorithme *K*-means a été choisi comme cas d'étude des algorithmes de *clustering* pour cette thèse pour diverses raisons en lien avec le contexte auquel nous nous intéressons qui est le ML dans l'EI.

1. La popularité : le *K-means* est l'un des algorithmes d'apprentissage les plus utilisés. Par exemple, dans [55], les auteurs recensent les algorithmes de *clustering* les plus utilisés dans le domaine de la recherche sur la pollution de l'air en prenant les travaux publiés entre 1980 et 2019. Ils montrent que le *K-means* a été utilisé dans 70%

1. La dimension fractale correspond au logarithme de nombre de cellules de dimension r nécessaires pour recouvrir les observation du *data-set* divisé par le logarithme de la dimension r de la cellule [8].

Catégorie	Algorithme	Complexité	Addaptabilité à un gros volume de données
(1)	*K*-means	Faible	Oui
	K-medoids	Elevée	Non
	PAM	Elevée	Non
	CLARA	Moyenne	Non
	CLARANS	Elevée	Non
(2)	BIRCH	Faible	Oui
	CURE	Faible	Oui
	ROCK	Elevée	Non
	Chameleon	Elevée	Non
(3)	FCM	Faible	Non
	FCS	Elevée	Non
	MM	Moyenne	Non
(4)	DBCLASD	Moyenne	Oui
	GMM	Elevée	Non
(5)	DBSCAN	Moyenne	Oui
	OPTICS	Moyenne	Oui
	Mean-shift	Elevée	Non
(6)	Click	Faible	Oui
(7)	CLIQUE	Faible	Non
(8)	FC	Faible	Oui
(9)	COBWEB	Faible	Oui

TABLE 2.1 – Comparatif des algorithmes de *Clustering* traditionnels. Adapté de [154]

des études considérés suivi des algorithmes de *clustering* hiérarchique. Dans [151], les auteurs classent le *K-means* comme l'algorithme de *clustering* le plus utilisé.

2. L'adéquation à un large volume de données : dans le contexte d'explosion du volume de données mondial [116], les volumes des *data-set* croissent de manière très rapide. Par conséquent, ce critère est important dans le choix d'algorithme de cas d'étude. Comme montré dans le tableau 2.1, le *K-means* est parmi les algorithmes qui conviennent à des volumes de données s'apparentant au *Big Data* [154].
3. La faible complexité temporelle[2] : dans [154], les auteurs expriment la complexité du *K-means* comme étant le produit du nombre de *clusters* par le nombre d'observations par le nombre d'itérations nécessaires à la convergence des *clusters*, ce qui est relativement bas comparés aux autres algorithmes qui présentent des complexités quadratiques en fonction du nombre d'observations. L'argument de faible complexité temporelle est donc en faveur du *K-means* et justifie sa popularité [55, 154]. Cet argument est d'autant plus important dans le contexte de l'EI, au vu des contraintes temps-réel qui peuvent exister sur certains systèmes embarqués (voir la section 2.3 page 53).

2.2.3 L'algorithme du *K-means*

La formulation du problème du *K-means*, traduit en *K*-moyennes a été proposée la première fois par Hugo Steinhaus [67] et qui est défini comme suit [7].

Définition 6. *Soit K le nombre de clusters à obtenir. Il s'agit de trouver l'affectation des observations à K clusters de manière à minimiser la variance intra-cluster donnée par :*

$$\underset{C_1,C_2,\dots,C_K}{\arg\min} \sum_{k=1}^{K} \|\vec{x} - \vec{\mu}_k\|_2^2 \tag{2.1}$$

Où $\|.\|_2$ représente la norme euclidienne détaillée dans la définition 7, C_i représente le cluster i et $\vec{\mu}_i$ représente le point central de ce cluster donné par l'équation suivante :

$$\vec{\mu}_i = \frac{1}{|C_i|} \sum_{\vec{x} \in C} \vec{x} \tag{2.2}$$

Définition 7. *La distance Euclidienne entre deux points x et y de dimension d est définie comme suit : $d(x,y) = \|\vec{x} - \vec{y}\|_2 = \sqrt{\sum_{i=0}^{d}(x_i - y_i)^2}$*

2. La complexité temporelle est une description (mise en équation) du temps nécessaire à l'exécution du problème en fonction des paramètres d'entrée.

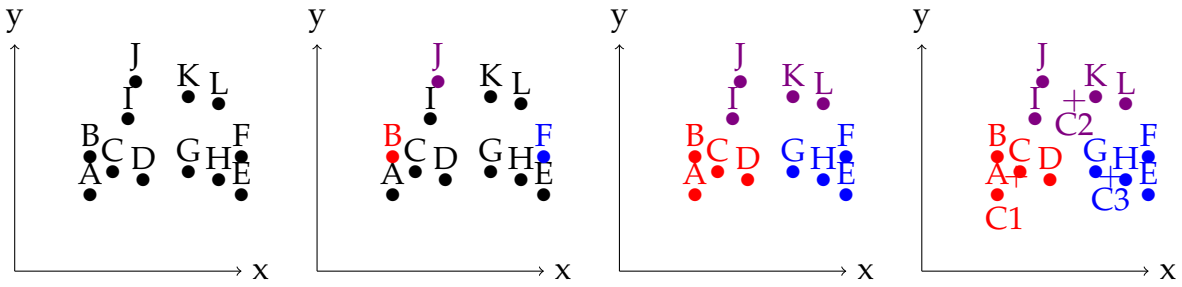


FIGURE 2.2 – Illustration du fonctionnement de l'algorithme *K*-means

Cependant, il est impossible de résoudre l'équation (2.1) de manière exacte. Stuart Lloyd propose alors une heuristique [83] qui permet de résoudre le problème.

L'heuristique est donné dans l'algorithme 1. La première étape consiste à initialiser les centres (ligne 1). Dans la littérature, cela est fait soit de manière aléatoire en échantillonnant K observations du *data-set* pour les considérer comme centres initiaux, ou suivant une heuristique d'initialisation [24] qui vise à maximiser la distance entre les centres initiaux. Par la suite, chaque observation du *data-set* est parcourue et affectée au centre le plus proche en terme de distance euclidienne (ligne 4). Une fois toutes les observations parcourues et affectées, les centres de *clusters* sont recalculés tel que chaque nouveau centre du *cluster* i est le barycentre des observations qui lui sont affectées selon l'équation (2.2). Le processus s'arrête lorsque la condition de convergence est atteinte. Cette dernière est caractérisée par le fait que les centres ne varient plus entre deux itérations consécutives, ou que le nombre d'itérations atteint un nombre maximal it_{max}.

NB : dans ce manuscrit, le nombre de *clusters* K à constituer est considéré comme une entrée. Néanmoins, il existe plusieurs méthodes pour déterminer le nombre K [74].

Algorithm 1: Pseudo Algorithme du *K*-means

```
   Data: D : Data-set
   Result: K centres
1  Initialiser les K centres
2  while Non convergence do
3      for i ← 0 to N − 1 do
4          Affecter l'observation i du data-set au centre le plus proche
5      Recalculer les centres selon la formule (2.2).
```

Dans la figure 2.2, on illustre le fonctionnement de l'algorithme sur un exemple de 12 éléments (A à L) où l'on veut définir 3 *clusters*. La première étape consiste à définir 3 centres initiaux B, F et J (dans cet exemple, nous avons volontairement choisi des centres

initiaux éloignés les uns par rapport aux autres afin d'accélérer la convergence). Ensuite, les autres éléments du *data-set* sont affectés au centre qui leur sont les plus proches. Les centres sont mis à jour en calculant le barycentre de chaque *cluster* (C_1, C_2 et C_3). Une nouvelle itération de l'algorithme est effectuée en affectant chaque point au centre qui lui est le plus proche. L'affectation restant la même, le calcul des centres donne les mêmes centres qu'à l'itération précédente ce qui signifie que la condition de convergence est atteinte.

2.2.4 Les algorithmes de classification

Dans cette section, nous décrivons les algorithmes classiques de classification. Nous mettons en évidence leur similarité et leurs différences.

1. **Les k-plus proches voisins (K-NN) :** il est considéré comme l'un des plus anciens algorithmes de classification. Dans cet algorithme, pour identifier la classe d'une nouvelle observation, il faut identifier les k observations du *data-set* qui lui sont les plus proches. La classe assignée à cette nouvelle observation est alors la classe majoritaire à laquelle appartiennent les k plus proches voisins [3] [68]. On qualifie souvent cet algorithme d'apprentissage paresseux (*Lazy Learning)* [7], car le calcul des k plus proches voisins se fait au moment de l'inférence, c'est à dire lorsqu'on connaît l'observation à classifier.

2. **La classification naïve bayésienne (NBC)** est un modèle d'apprentissage probabiliste basé sur la loi de Bayes [99]. Elle consiste à calculer la probabilité pour une observation d'appartenir aux différentes classes du problème connaissant les valeurs de ses propriétés, puis de choisir la classe pour laquelle la probabilité est maximale. Plus formellement, soit $\vec{x}$ une observation et Y la classe effective de cette observation (inconnue que l'on cherche), alors : $Y = \arg\max_{c=1,..,C} P(Y = c|X = \vec{x})$. Selon la loi de Bayes :

$$P(Y = c|X = \vec{x}) = \frac{P(Y = c)P(\vec{x}|Y = c)}{P(\vec{x})} \tag{2.3}$$

   Cette méthode repose néanmoins sur une hypothèse forte, qui est que les propriétés qui caractérisent le *data-set* soient indépendantes.

3. **La Machine à Vecteurs de Support (SVM) :** c'est un modèle d'apprentissage statistique [114]. Le principe de cette méthode consiste à utiliser des hyperplans pour séparer les observations des différentes classes [7]. La recherche des hyperplans consiste à résoudre une série de problèmes d'optimisation quadratiques convexes.

3. À ne pas confondre avec le K du K-means qui désigne le nombre de *clusters*. Ici, il est utilisé pour dénombrer les voisins que l'on considère.

4. **Les réseaux de neurones, ou *Neural Network*(NN) :** selon [7], ils sont basés sur le modèle de perceptron qui est un modèle de classification linéaire. Le perceptron est formé d'une couche d'entrée de d neurones qui correspondent chacun à une propriété du *data-set* et un neurone de biais, et une couche de sortie composée d'un neurone (un seul dans le cas d'une classification binaire, n dans le cas d'une classification à n classes) qui calcule la combinaison linéaire des neurones de la couche d'entrée. Pour une observation $\vec{x}$ du *data-set* sur laquelle on applique le perceptron, on fait correspondre chaque neurone de la couche d'entrée à une composante de $\vec{x}$. Au niveau de la couche de sortie, le neurone calcule la combinaison linéaire des composantes de $\vec{x}$ selon l'équation suivante :

$$o(\vec{x}) = w_0 + \sum_{j=1}^{d} w_j \cdot x_j \tag{2.4}$$

   Le neurone applique ensuite une fonction d'activation qui permet de retourner 1 si l'observation appartient à la classe que le neurone de sortie reconnaît et 0 sinon. L'opération d'entraînement consiste à trouver les bons poids w_j de manière à prédire l'appartenance à une classe. La modélisation d'un problème de classification sous forme de perceptron est cependant souvent impossible car il s'agit d'une modélisation linéaire, donc insuffisante pour traduire la complexité des problèmes. Le perceptron multicouche, est alors proposé pour répondre à cela. Il consiste à superposer des couches intermédiaires (couches cachées) entre les couches d'entrée et de sortie.

5. **Les arbres de décision :** les arbres de décision sont des classifieurs très populaires [99]. Un arbre de décision est un graphe arborescent dont le nœud racine contient toutes les observations du *data-set*, les nœuds internes sont des conditions basées sur les propriétés du *data-set*, les nœuds feuilles contiennent un sous-ensemble des observations du *data-set* et sont étiquetés par la classe prédominante parmi les observations de ce sous-ensemble. Ainsi, chaque branche de l'arbre est une succession de conditions sur les valeurs des propriétés de l'arbre qui permettent de discriminer les classes possibles des observations. La définition d'un arbre de décision est donnée de manière plus formelle dans la section 2.2.5. L'inconvénient de cet algorithme est qu'il est sujet au problème de sur-apprentissage expliqué dans la définition 8 page 48.

6. **Les modèles d'apprentissage ensemblistes :** les classifieurs énumérés précédemment sont sujet aux erreurs de classification. Ces dernières englobent

deux composantes : le biais, qui peut être défini comme la précision des prédictions faites par le modèle ; et la variance définie comme la précision du classifieur quand il est entraîné sur plusieurs *data-sets*. Il existe un compromis en biais et variance. En effet, lorsque l'on réduit le biais, ce qui revient à entraîner le classifieur pour qu'il corresponde mieux aux observations du *data-set*, le risque d'avoir une classification erronée avec une nouvelle observation est élevée, ce qui signifie que la variance augmente, et vice-versa. L'approche des modèles d'apprentissage ensemblistes consiste à construire plusieurs classifieurs à faible biais puis de les combiner (par plusieurs méthodes) afin de réduire la variance. Il existe plusieurs stratégies d'apprentissage ensembliste :

— Le *bagging* [4][19] : il consiste à entraîner T classifieurs indépendants où chacun de ces classifieurs est entraîné sur un échantillon avec remise de N observations du *data-set* de départ, les résultats des classifications sont combinés au moment de l'inférence en effectuant un vote majoritaire. Les RANDOM FORESTS qui seront détaillées dans la section qui suit sont dans cette catégorie ;

— Le *boosting*[127] : à la différence du *bagging*, les classifieurs sont construits de manière obligatoirement itérative afin d'augmenter le poids des observations mal classifiées par le classifieur i dans la constitution du *bootstrap* du classifieur $i+1$;

— Le *stacking* : cela consiste à entraîner les classifieurs de manière indépendante. En revanche, la combinaison des résultats ne se fait pas par vote majoritaire, mais en entraînant un modèle afin qu'il prédise quels seront les classifieurs performants pour une observation donnée ;

— Le mélange d'experts (*mixture of experts*) : dans cette méthode, la tâche de classification est divisée en plusieurs sous-tâches. Pour chacune de ces dernières, un réseau de neurones (classifieur ou expert) est entraîné indépendamment. Par la suite, un « réseau de portes » qui a pour but de pondérer les sorties des différents classifieurs (réseaux de neurones) est entraîné sur les observations du *data-set* de sorte à attribuer des poids plus importants, aux classifieurs les plus précis pour une observation donnée.

4. Ce mot résulte de la combinaison des deux mots « *Bootstrap* » qui représente l'ensemble des observations formé par échantillonnage avec remise sur le *data-set*, et « *Aggregation* » qui est l'opération de combinaison des résultats.

Méthode	Avantages	Inconvénients
K-NN	Facile à mettre en œuvre.	Requiert le stockage du *data-set* pendant la phase d'inférence ; Complexité temporelle élevée.
NBC	Facile à mettre en œuvre ; Facile à interpréter.	L'hypothèse d'indépendance entre les propriétés du *data-set* n'est souvent pas réaliste.
SVM	Précision élevée.	Complexité temporelle élevée ; Difficile à concevoir pour les cas de classification multiclasses.
ANN	Précision élevée ;	Complexité temporelle élevée ; Difficile à interpréter.
DT	Facile à interpréter ; Complexité faible.	Sujet au problème d'*overfitting*.

TABLE 2.2 – Comparatif des méthodes de classification, adapté de [114]

Le tableau 2.2 synthétise les avantages et inconvénients des méthodes ci-dessus.

Définition 8. *L'**overfitting** ou le surapprentissage est le phénomène qui consiste à obtenir, à l'issue de la phase d'apprentissage, un modèle qui n'est pas généralisable et qui est « trop » adapté aux observations du data-set. En effet, pour obtenir un modèle qui prédit parfaitement la classe des observations du data-set, il suffirait d'énumérer les propriétés des observations de chaque classe. Néanmoins, si l'on confronte ce modèle à une nouvelle observation (distincte de toutes les observations du data-set), le modèle sera incapable de prédire sa classe, car trop spécifique au data-set [47].*

Justificatif du choix des Random Forests comme algorithme de classification

Notre choix de cas d'étude pour un algorithme de classification s'est orienté vers les RF pour les raisons suivantes :

~~1.~~ La popularité : parmi les algorithmes de classification, les RF font partie des plus utilisés [82, 54] ;

2. La précision : dans [73], plusieurs modèles d'apprentissage précédemment définis sont évalués en terme de précision : les *K*-NN, les DT, les SVM, et les RF. Les RF ont donné la meilleure précision sur 5 des 8 *data-sets* testés. De plus, les RF ne sont pas sujet au problème de l'*overfitting* [54] ;

3. La faible complexité temporelle : le temps d'entraînement des RF est rapide en comparaison des autres modèles de classification [158]. Selon [54], les RF sont plus rapides à entraîner que les réseaux de neurones convolutionnels. Par ailleurs, l'entraînement est facilement parallélisable du fait qu'il s'agisse de plusieurs arbres construits de manière indépendante [158] ;

4. La simplicité : l'apprentissage d'un arbre de décision ne requiert qu'un seul paramètre en entrée, qui est le nombre d'arbres T [158].

2.2.5 Arbres de décision et Random Forests

Une Random Forests ou forêt aléatoire est constituée d'un ensemble d'arbres de décision qui représentent chacun un classifieur indépendant. Dans ce qui suit, nous définissons un arbre de décision, nous donnons son algorithme de construction, et enfin, nous expliquons le principe d'inférence sur une RF.

Arbre de décision :

Définition 9. *Un arbre de décision est un graphe arborescent orienté [121]. L'ensemble des nœuds se divise en trois catégories : le nœud racine (il n'a pas d'ascendant), les nœuds internes (qui ont des descendants, et un ascendant) et les nœuds terminaux ou feuilles (qui n'ont pas de descendants). Chaque nœud interne représente un test qui divise l'espace d'observations selon une propriété du data-set [49]. Chaque nœud feuille est étiqueté selon la classe majoritaire du sous-ensemble d'observations qui répondent aux tests de l'ensemble des nœuds de la branche allant du nœud racine au nœud feuille en question.*

Définition 10. *Le **bootstrapping** est une méthode statistique dans laquelle l'ensemble d'observations qui vont constituer le **bootstrap** est formé. Étant donné un data-set de N observations, le bootstrap est formé en tirant aléatoirement, avec remplacement le data-set. Par conséquent, à chaque tirage, chaque observation a une probabilité de $1/N$ d'être tirée.*

Algorithm 2: pseudo algorithme de construction d'un arbre de décision [158]

```
   Data: D : Data-set
   Result: Arbre de décision
1  Création d'un bootstrap
2  while il existe un nœud impure n do
3  |  Création d'un sous-ensemble aléatoire de propriétés F
4  |  for f ← 0 to |F| − 1 do
5  |  |  Tentative de division du nœud n en deux nœuds enfants selon la propriété f
6  |  Choix de la meilleure propriété f* et création effective des nœuds enfants
```

La construction d'un arbre de décision consiste alors à trouver les bons tests qui permettent d'obtenir les nœuds les plus pures possibles selon le critère du *Gini Index* donné dans la définition 11.

Dans l'algorithme 2, nous montrons le pseudo code de construction d'un arbre de décision. La première étape consiste à créer un *bootstrap* qui est un ensemble d'observations tirées par un échantillonnage avec remise sur le *data-set* et d'affecter les observations de ce *bootstrap* au nœud racine (ligne 1 de l'algorithme 2). L'étape suivante consiste à créer un sous-ensemble aléatoire de propriétés du *data-set* F, dont le nombre d'éléments [5] est $|F| = \sqrt{d}$ (ligne 3 de l'algorithme 2). L'étape suivante consiste à tenter de diviser le nœud courant selon chaque propriété de l'ensemble F de manière à obtenir les nœuds enfants les plus pures possibles (voir la définition 11) (lignes 4-5 de l'algorithme 2). La meilleure

5. Le nombre d'éléments dans l'ensemble F a été défini comme $\sqrt{d}$ pour les arbres de décision pour un problème de classification et $d/3$ pour un problème de régression [158].

propriété f^* qui donne les nœuds enfants les plus pures est choisie pour la division effective du nœud (ligne 6 de l'algorithme 2). Ce processus s'arrête lorsque tous les nœuds enfants sont pures, ou lorsque l'arbre a atteint la profondeur maximale [6].

Pour un problème de classification, le critère utilisé pour choisir la meilleure propriété de division est le **Gini Index** défini ci-après.

Définition 11. *Soit un problème de classification de K classes notées* $1, ..., K$*, pour un nœud donné, on définit le Gini index comme :*

$$Q = \sum_{k \neq k'}^{K} \hat{p_k} \cdot \hat{p_{k'}} \tag{2.5}$$

tel que $\hat{p}_k$ *est la proportion des observations du nœud de classe k :*

$$\hat{p}_k = \frac{1}{n} \sum_{i=1}^{n} I(y_i = k) \tag{2.6}$$

où

$$I(y_i = k) = \begin{cases} 1 & \text{si } y_i \text{ est de classe } k \\ 0 & \text{sinon} \end{cases}$$

Une division candidate crée deux nœuds enfants, un nœud gauche et un nœud droit, dont les indexes Gini sont respectivement Q_g *et* Q_d*, et les nombres d'éléments respectifs* n_g *et* n_d*. L'index Gini de la division candidate associé à une propriété* f *est alors :*

$$Q_f = n_g \cdot Q_g + n_d \cdot Q_d \tag{2.7}$$

La propriété qui donne la valeur de Gini index minimale est choisie pour la division du nœud (le calcul du Gini Index est illustré dans l'exemple qui va suivre).

Afin d'illustrer le fonctionnement de cet algorithme, nous utilisons un exemple simple de construction d'arbre de décision basé sur le *data-set* donné dans le tableau 2.3 composé de 8 observations caractérisées par 4 propriétés (f_1, f_2, f_3 et f_4). La figure 2.3 illustre ce fonctionnement. La première étape **création du *bootstrap*** constitue l'échantillon sur lequel sera construit l'arbre de décision, ici les éléments échantillonnés sont $\{A, C, E, C, H, A, B, F\}$. L'étape suivante consiste à effectuer une **tentative de division** selon un sous-ensemble de propriétés. Comme la dimension du problème est de 4, le nombre de propriétés à tester est 2, dans cet exemple, il s'agit de $\{f_1, f_4\}$. On obtient alors deux divisions potentielles. Afin de choisir la meilleure propriété, l'index Gini est utilisé :

— Pour la division du nœud N_0 :

6. La profondeur maximale est définie par le concepteur du modèle.

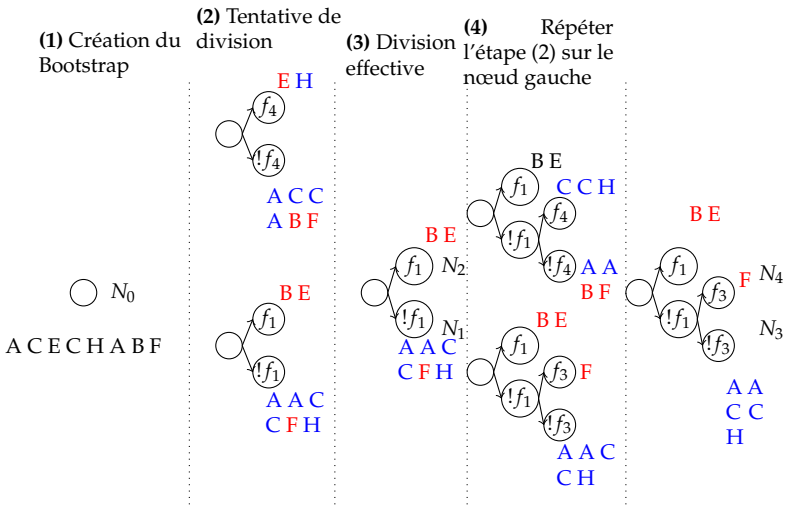


FIGURE 2.3 – Construction d'un arbre de décision

— La tentative de division selon f_1 : le nœud gauche regroupe 5 observations de la classe 0 et 2 observations de la classe 1. Par conséquent : $Q_g = 2 \cdot 1/6 \cdot 5/6 = 5/18$.
Le nœud droit contient deux éléments de la classe 1. Donc, $Q_d = 2 \cdot 2/2 \cdot 0$.
L'index Gini du nœud N_0 lorsque divisé selon f_1 est alors : $Q_{f_1} = 0 + 6 \cdot 5/18 = 5/3$.

— La tentative de division selon f_4 : le nœud fils gauche regroupe 4 observations de la classe 0 et 2 observations de la classe 1. Donc, $Q_g = 2 \cdot 2/6 \cdot 4/6 = 4/9$.
Le nœud fils droit contient une observations de classe 0 et une observation de classe 1. $Q_d = 2 \cdot 1/2 \cdot 1/2$.
L'index Gini selon du nœud N_0 selon la propriété f_4 est alors : $Q_{f_4} = 2 \cdot 1/2 + 6 \cdot 4/9 = 11/3$.

— Par conséquent, f_1 est choisi pour la division du nœud N_0. La **division effective** a alors lieu selon f_1.

Le même procédé est appliqué sur les nœuds résultants. Le nœud N_2 est pure puisque les observations qui lui sont affectées sont toutes les deux de la même classe. Le nœud N_1 n'est, quant à lui, pas totalement pure, le même procédé est appliqué sur ce nœud. Les propriétés choisies pour la tentative de division sont $\{f_3, f_4\}$. f_3 donne la meilleure division et est utilisée pour la division effective. Tous les nœuds feuilles obtenus sont pures ; l'arbre de décision est alors entièrement construit.

Label	f_1	f_2	f_3	f_4	Classe
A	0	0	0	0	0
B	1	0	1	0	1
C	0	1	0	1	0
D	0	0	0	0	0
E	1	1	1	1	1
F	0	1	1	0	1
G	1	0	0	1	0
H	0	1	0	1	0

TABLE 2.3 – Exemple de *data-set*

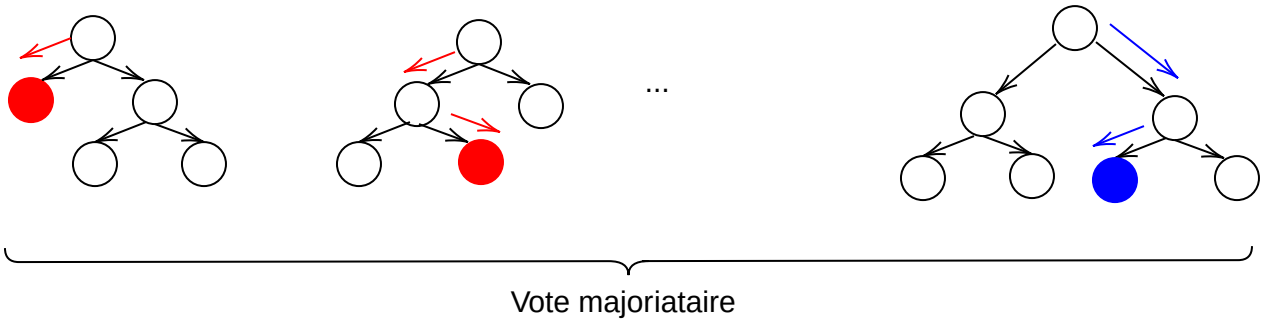


FIGURE 2.4 – Étape d'inférence sur une forêt aléatoire

Forêts aléatoires

La forêt aléatoire est un modèle de classification ensembliste. Il consiste à construire un ensemble de T arbres de décision selon l'algorithme 2. L'inférence de la classe d'une nouvelle observation se fait par vote majoritaire sur les classes prédites par les arbres de la forêt. La figure 2.4 montre la procédure d'inférence de la classe (bleu ou rouge) d'une observation $\vec{x}$. L'observation $\vec{x}$ est classifiée selon chaque arbre de décision (rouge pour l'arbre 1, rouge pour l'arbre 2, ..., bleu pour l'arbre T). La classe retournée est la classe majoritaire parmi les classes retournées par les arbres de décision.

2.2.6 Conclusion

Dans cette deuxième partie du chapitre, nous avons décrit l'une des trois composantes clé de l'EI : les algorithmes de ML. Nous avons introduit et passé en revue la taxonomie de ces algorithmes. Nous nous sommes focalisée sur les algorithmes de *clustering* et de classification et avons choisi un algorithme dans chacune de ces deux catégories comme

cas d'étude pour cette thèse. Pour le *clustering*, nous avons choisi l'algorithme du *K-means*, pour sa rapidité, sa popularité et son adéquation avec de large volumes de données. Pour la classification, nous avons opté pour les Random Forests, pour leur popularité, précision, rapidité et simplicité. Dans ce qui suit, nous décrivons les défis soulevés par l'EI, suivi d'une description de la contrainte mémoire et des travaux de recherche s'y rapportant dans l'EI, qui fait l'objet de cette thèse.

2.3 Les défis de l'*Edge* intelligence

Dans la section 2.1, nous avons défini le domaine émergent qu'est l'EI ainsi que les principaux axes de recherche qui sont déjà investis. Nous avons vu dans la section précédente que l'EI repose sur un large spectre d'algorithmes de ML. Dans cette section, nous passons en revue les principaux défis que soulève l'EI en nous basant sur une revue de la bibliographie [18, 162, 153, 161].

— **L'empreinte mémoire.** L'un des plus grands défis auquel fait face l'EI est la réduction de l'empreinte mémoire sur les dispositifs embarqués [18, 161]. En effet, la plupart des algorithmes d'entraînement de modèles intelligents agissent sur de gros volumes de données brutes et auxiliaires générées à l'issue de calculs intermédiaires, qui occupent un espace mémoire important. Au vu de la limitation de la capacité mémoire sur les dispositifs embarqués, il est nécessaire de réduire l'empreinte mémoire des algorithmes d'entraînement [18].

— **Le temps d'exécution.** Toute application doit s'exécuter dans un temps compatible avec son usage. Il est, donc, important d'optimiser les algorithmes d'entraînements de sorte à réduire le temps d'exécution. Par ailleurs, les composants électroniques des dispositifs embarqués, à la différence d'un serveur *cloud* par exemple, ne sont pas conçus pour l'exécution continue d'une tâche pendant plusieurs heures [161]. Dans [18], les auteurs mentionnent plusieurs leviers sur lesquels il faut agir afin de réduire le temps d'exécution : les opérations arithmétiques, les multiplications et divisions sont à titre d'exemple plus coûteuses que des additions ; les instructions de branchement, l'allocation de mémoire virtuelle et les transferts de données.

— **La consommation énergétique.** Sur les dispositifs alimentés par batteries, la consommation énergétique est l'une des contraintes les plus importantes. La communication est parmi les tâches les plus énergivores. Dans le cas des objets connectés, elle représente 70% de la consommation totale d'énergie [18]. L'architecture EI décentralisée telle que décrite dans la section 2.1.2, permet de s'affranchir du coût énergétique de la communication. Un autre moyen de réduire la consomma-

tion énergétique est d'agir sur le code des algorithmes d'entraînement, en réduisant les calculs et opérations non nécessaires. Cela réduit le temps où le dispositif est en état « actif » et où la consommation énergétique est maximale. Le type de mémoire utilisée sur ces dispositifs agit également sur la consommation énergétique, de part les coûts des opérations de lecture, écriture, rafraîchissement des données qui varient d'une technologie mémoire à une autre (voir la section 2.4 page 59).

— **La précision et la qualité des données collectées.** La précision du modèle entraîné dépend du *data-set* sur lequel il a été entraîné [18, 153]. En effet, il doit contenir suffisamment d'observations de manière à former un modèle générique. Dans le contexte de l'EI décentralisée, les données collectées sont souvent insuffisantes [161] et nécessitent l'utilisation de différentes techniques, par exemple, l'augmentation des données, afin pouvoir de entraîner le modèle [153]. Outre le nombre d'observations du *data-set*, la qualité des données collectées a également une influence sur la précision du modèle.

— **La sécurité et la confidentialité.** La sécurité du système peut être dégradée lorsque le fonctionnement de celui-ci dépend du modèle intelligent appliqué. Par exemple, si le modèle peut être hacké en utilisant des données qui ne correspondent pas aux entrées attendues (bruits, types incompatibles avec les entrées), le système peut se retrouver dans des scénarios dangereux ou être endommagé. Par ailleurs, la confidentialité des données collectées (personnelles, militaires, etc) sur les dispositifs embarqués est un point critique dans le cadre de l'EI [18].

— **La scalabilité et la flexibilité.** En ML, un modèle entraîné sur un *data-set* collecté sur une plateforme donnée peut ne pas être applicable sur une autre plateforme, où la donnée est collectée différemment dû à l'hétérogénéité des capteurs et/ou des environnements de collecte par exemple [161] . Par conséquent, la standardisation des modèles et des données est importante pour assurer la flexibilité dans l'EI.

2.4 La mémoire et le stockage dans le contexte de l'*Edge* Intelligence

Dans cette thèse, nous nous intéressons spécifiquement à la contrainte du manque de ressources de mémorisation et de stockage sur les dispositifs embarqués, lorsqu'il s'agit d'apprentissage dans une architecture d'EI décentralisée. Plus précisément, la problématique laquelle nous nous intéressons est celle des mouvements de données entre la mémoire principale et l'espace de stockage, lorsque la mémoire principale ne suffit pas

à contenir le volume de données sur lequel l'entraînement du modèle doit se faire. En effet, ces mouvements sont coûteux en temps, étant données les différences de performance des technologies de mémorisation et de stockage.

Afin de mieux cerner cette problématique, nous structurons cette section de la manière suivante : nous commençons par décrire succinctement la hiérarchie mémoire et les différences de performances entre les technologies existantes. Nous décrivons, par la suite, la gestion de l'espace d'adressage d'une application par le système d'exploitation, pour comprendre les communications qui s'opèrent entre les différents niveaux de la hiérarchie mémoire. Ensuite, nous effectuerons une synthèse des solutions proposées dans la littérature pour la réduction de l'empreinte mémoire et de la communication entre niveaux de la hiérarchie mémoire d'une application de manière générale, et dans le cas de des applications d'EI en particulier.

2.4.1 Généralités sur les dispositifs mémoire et leur gestion par les systèmes d'exploitation

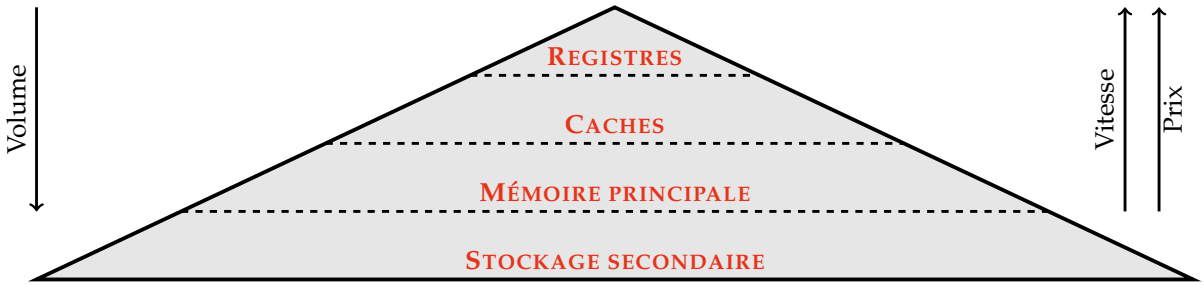


FIGURE 2.5 – Hiérarchie des mémoires, adaptée de [16]

La hiérarchie mémoire dans les systèmes informatiques

La figure 2.5 illustre la hiérarchie mémoire dans les systèmes informatiques. Elle se divise en quatre niveaux selon la distance du dispositif de mémorisation par rapport à l'unité de calcul (CPU). Plus le dispositif est performant en terme de bande-passante et d'endurance, plus il est proche du CPU [130]. Les quatre niveaux montrés dans la figure sont décrits ci-après :

1. Les registres : ils constituent une très petite quantité mémoire intégrée directement au niveau du CPU afin d'accélérer l'accès aux données fréquemment utilisées. Ils sont généralement mis en œuvre en utilisant la technologie *Static Random Access*

Memory (SRAM) qui est considérée comme très rapide par rapport aux autres technologies mémoire (voir la section 2.4 page 59).

2. La mémoire cache : son objectif est de réduire l'écart de vitesse entre le CPU et la mémoire principale. Il repose sur le principe de localité[7] pour stocker les données qui sont susceptibles d'être utilisées dans un future proche. La technologie utilisée pour mettre en œuvre les caches est la SRAM. La capacité du cache est limitée pour deux raisons : la coût au méga-octets de SRAM est élevé ($1-100) [141] ; et la taille d'une cellule SRAM est grande par rapport aux autres technologies mémoire [16] (voir la section 2.4 page 59).
3. La mémoire principale : les données qu'elle contient sont accessibles par octets. Classiquement, elle repose sur la technologie DRAM qui est moins chère comparée à la SRAM [16]. La DRAM est, néanmoins, relativement lente par rapport au CPU. En effet, le temps d'accès à une donnée sur la DRAM est de 10 à 100 ns [141], ce qui est élevé étant donné les performances des processeurs modernes. D'autres technologies sont envisagées comme alternatives ou extensions à la DRAM, comme les mémoires non-volatiles (NVM) qui seront abordées dans la section 2.4.
4. Le stockage secondaire : il assure le stockage permanent des données dans les systèmes informatiques. Il repose sur diverses technologies : historiquement, la bande magnétique, le disque dur, puis la mémoire flash. Ces technologies assurent une accessibilité par blocs, mais sont toutes plus lentes que les technologies utilisables comme mémoire principale et cache. Le stockage secondaire permet également de fournir aux systèmes informatiques une mémoire de travail (principale) bien plus grande que la mémoire disponible, et à bas coût grâce au mécanisme de *swap* supporté par certains systèmes d'exploitation [66]. Le mécanisme de *swap* sera décrit dans la section 2.4.1 page 59.

Les technologies mémoire existantes

À travers le temps, les technologies mémoire et leurs techniques d'intégration sur les systèmes d'informatiques ont évolué. Le tableau 2.4 dresse un comparatifs des technologies existantes selon les critères suivants :

— **La taille de la cellule :** la taille de la cellule dépend de la structure des matériaux

7. Il existe deux types de localités : (1) la localité spatiale [141], qui suppose que du fait de la structure cyclique des programmes et l'organisation des données en lignes (un nombre donné d'octets écris en mémoire à des adresses contiguës), les lignes proches de la ligne la plus récemment utilisée ont une forte probabilité d'être utilisées dans un future proche ; (2) la localité temporelle [141], qui suppose qu'une ligne utilisée a une forte probabilité d'être réutilisée dans un future proche.

qui la compose. Ce critère a une influence directe sur la capacité de stockage des technologies mémoire. En effet, plus la taille de la cellule est petite plus la densité de stockage, c'est-à-dire le nombre de cellules pouvant être intégrées sur une même surface, sera grande. Une densité élevée est intéressante pour réduire l'espace occupé par la puce mémoire pour un dispositif embarqué par exemple. En revanche, une densité élevée augmente les courants de fuite et par conséquent la consommation statique de la mémoire [107].

— **L'endurance :** cette propriété traduit le seuil de nombre d'écritures que peut supporter une cellule mémoire. Au-delà de ce seuil, la cellule ne retient plus de donnée [16].

— **La latence des accès :** une technologie mémoire est caractérisée par les temps d'accès en lecture et en écriture d'une donnée.

— **La consommation énergétique :** elle dépend également des composants des cellules de la mémoire. Elle se décompose en deux parties : la consommation statique, qui est importante lorsque la technologie mémoire est volatile et nécessite un rafraîchissement périodique des données. La deuxième partie de la consommation énergétique est la consommation dynamique liée aux opérations de lecture et écriture.

Dans le tableau 2.4, nous reprenons quelques technologies mémoire traditionnelles, et des mémoires non volatiles dites émergentes. Dans ce qui suit, nous reprenons cette catégorisation et plaçons chacune des technologies dans le niveau où elle est communément intégrée dans la hiérarchie mémoire donnée dans la figure 2.5.

— **Les technologies mémoire classiques :** elles sont, aujourd'hui, largement utilisées sur les systèmes informatiques. Les technologies montrées dans le tableau sont :

1. La SRAM : c'est la technologie la plus répandue pour la mémoire cache [107]. En effet, le rôle du cache est de garantir un temps d'accès très court à la donné par le CPU. Au vu des latences de la SRAM comparée aux autres technologies, elle est appropriée à la mise en œuvre du cache [107].
2. La DRAM : cette technologie est largement utilisée pour la mise en œuvre de la mémoire principale. Elle a pour avantage des latences en lecture et écriture relativement réduites ainsi qu'un coût par Mo réduit. En revanche, du fait de sa volatilité, elle nécessite un rafraîchissement périodique des données, ce qui augmente sa consommation statique, et réduit rend les données un disponibles pendant l'opération de rafraîchissement.
3. La NAND flash : elle s'est imposée comme technologie de stockage depuis plusieurs années [107, 100], notamment sur les dispositifs embarqués. Parmi

ses avantages, le fait que la latence d'accès en lecture ne dépende pas de la position de la donnée par rapport à l'accès précédent, contrairement au cas d'un disque dur HDD. Par conséquent, les coûts des lectures séquentielles et aléatoires sont équivalents [16]. Elle présente, néanmoins, une asymétrie en latence et en énergie entre l'opération de lecture et d'écriture, cette dernière étant plus lente et énergivore que la lecture, ainsi qu'une endurance relativement faible par rapport aux autres technologies.

— **Les mémoires non volatiles émergentes :** ces mémoires sont très prometteuses principalement grâce à la propriété de non volatilité. Cette dernière repose sur des phénomènes physiques inhérents au matériau qui compose la cellule mémoire, et qui lui confère l'état de conduction ou de résistivité (ce qui correspond à un bit à 1 ou à 0) [107]. De ce fait, la cellule n'a pas besoin d'être alimentée en continu, d'où leur basse consommation statique. Outre la non volatilité, elles ont pour avantage une forte densité puisque leurs cellules ont des tailles réduites comparées à la SRAM et la DRAM. En revanche, elles ont les mêmes inconvénients que la NAND flash, à savoir, une asymétrie en latence et en consommation énergétique entre la lecture et l'écriture, ainsi qu'une faible endurance. Nous expliquons ci-après quelques technologies NVM prometteuses :
  — La mémoire à changement de phase, ou *Phase Change Memory* (PCM) : cette technologie est basées sur un matériau, dit à changement de phase, le chalcogénure GST qui est caractérisé par la capacité de passer d'un état de haute résistivité à un état de faible résistivité selon la température qui lui est appliquée.
  — La mémoire ferro-électrique, ou *Ferroelectric Random Access Memory* (FeRAM) : une cellule FeRAM est constituée d'un transistor MOS et d'un condensateur basé sur le matériau *Lead Zirconate Titanate* qui conserve sa polarité même en l'absence d'alimentation électrique.
  — La mémoire résistives, ou *Resistive Random Access Memory* (ReRAM) : cette technologie est basée sur les *Memristors*, dont la résistivité dépend de la magnitude et de la polarité de l'impulsion à laquelle ils sont exposés.
  — La mémoire magnéto-résistives, ou *Magnetic Random Access Mmeory* (MRAM) : la cellule MRAM est composée d'un transistor MOS et d'un condensateur de type *Magnetic Tunnel Junction*.

  Leurs intégrations dans la hiérarchie mémoire est envisagée à plusieurs niveaux :

  1. Niveau du cache : pour ce niveau, la MRAM est la plus explorée dans la littérature [16]. La ReRAM est également envisagée mais pose un problème en terme d'endurance.

Technologie	SRAM	DRAM	HDD	NAND Flash	PCM	FeRAM	ReRAM
Taille de cellule (F^2)	120 -200	60-100	N/A	4-6	4-12	6-40	4-10
Endurance	10^{16}	$> 10^{15}$	$> 10^{15}$	$10^4 - 10^5$	$10^8 - 10^9$	$10^{14} - 10^{15}$	$10^8 - 10^{11}$
Latence en lecture	$0.2 - 2ns$	$10ns$	$3 - 5ms$	$15 - 35\mu s$	$20 - 60ns$	$20 - 80ns$	$10ns$
Latence en écriture	$0.2 - 2ns$	$10ns$	$3 - 5ms$	$200 - 500\mu s$	$20 - 150ns$	$50 - 75ns$	$50ns$
Consommation statique	élevée	moyenne	N/A	basse	basse	basse	basse
Consommation en lecture	basse	moyenne	N/A	basse	moyenne	basse	basse
Consommation écriture	basse	moyenne	N/A	basse	élevée	élevée	élevée

TABLE 2.4 – Caractéristiques des technologies mémoires [16] [141]

2. Niveau de la mémoire principale : les plus envisagées sont la PCM et la ReRAM. L'intégration la plus probable est une intégration horizontale, c'est-à-dire que la NVM représentera une extension de la DRAM plus dense et moins énergivore [16]. Elles sont également étudiées pour remplacer la DRAM.

3. Niveau du stockage : pour ce niveau, toutes les technologies NVM sont envisagées.

Pour la suite de ce manuscrit, nous considérons un dispositif embarqué dont la mémoire principale est basée sur la technologie DRAM. La mémoire secondaire est quant à elle basée sur la technologie NAND Flash. D'après le comparatif de performances donné dans le tableau 2.4, il existe un facteur moyen de 2500 entre la latence en lecture de la NAND Flash et la DRAM. Cet écart de performances a des conséquence sur les temps d'apprentissage des algorithmes de ML comme il sera démontré dans les analyses des cas d'études choisis (chapitres 3 et 4).

Gestion de l'espace d'adressage d'une application par le système d'exploitation

Nous nous intéressons à la gestion des espaces d'adressages des applications par le système d'exploitation, afin de comprendre le comportement de ce dernier lorsque le volume des données manipulées par une application est plus grand que l'espace disponible en mémoire principale. La description qui va suivre correspond à la gestion suivant le noyau Linux et est adaptée de [17].

Espace d'adressage d'un processus : l'espace d'adressage d'un processus est l'ensemble des adresses linéaires que le processus est autorisé à utiliser. Cet ensemble est distinct d'un processus à un autre. Le noyau Linux représente un intervalle d'adresses linéaires par « une région mémoire » caractérisée par une adresse de début, une longueur, toute deux multiples de 4096, ainsi que des droits d'accès.

La pagination à la demande : elle désigne une technique dynamique d'allocation mémoire qui consiste à différer l'allocation d'une page au plus tard, précisément, au moment où le processus tente d'accéder une adresse non présente en mémoire causant ainsi un défaut de page. Cette technique permet une meilleure utilisation de l'espace mémoire disponible en maintenant un nombre moyen élevé de pages libres en mémoire. Les traitements des défauts de page sont coûteux en temps. Cependant, leur coût est contre balancé par le fait qu'une fois le processus actif avec un sous-ensemble des pages appartenant à son espace d'adressage en mémoire, la probabilité qu'il demande à accéder à d'autres pages est basse du fait du principe de localité. Cette technique permet, donc, un compromis entre espace mémoire disponible et temps d'accès.

Une page qui est demandée par un processus peut être absente de la mémoire pour trois raisons :

1. Elle n'a jamais été référencée par le processus ;
2. Elle appartient au *mapping* non-linéaire d'un fichier : où les pages mémoire ne mappent pas des pages séquentielles du fichier, mais aléatoires ;
3. La page a déjà été accédée, mais est temporairement sauvegardée sur l'espace de stockage secondaire, au niveau de l'espace *swap*, afin d'augmenter le nombre de pages libres en mémoire.

Dans cette thèse, nous nous intéressons particulièrement au cas (3). En effet, dans le cas d'une application d'entraînement de modèle de ML, ce scénario est prompt à se produire, étant donné le volume du *data-set*. Nous allons décrire dans ce qui suit, ce qui amène le noyau à effectuer la sauvegarde d'une page sur l'espace *swap*, les étapes de l'opération de sauvegarde, ou mouvement d'une page mémoire vers l'espace *swap* (*swap-out*), ainsi que l'opération de récupération en mémoire d'une page *swappée* (*swap-in*).

Le mécanisme de *swap*. Le *swap* est un espace réservé sur le stockage secondaire qui sert de « *back-up* » (extension) à la mémoire principale lorsque l'espace physique de la mémoire ne suffit pas à contenir les pages des processus en cours d'exécution. Il est géré par le système d'exploitation de manière transparente pour les programmes. Bien que ce mécanisme représente une solution lorsque les besoins en mémoire excèdent l'espace physique disponible, les performances s'en trouvent dégradées à cause de la lenteur d'un accès au stockage secondaire par rapport à un accès à la mémoire.

L'espace *swap* est mis en œuvre comme partition sur le stockage secondaire, ou comme un fichier dans une partition. Dans un système, il peut y avoir plusieurs espaces *swap*. Chacun d'entre eux est une séquence, de pages de 4096 octets chacune par défaut.

Le mouvement des pages entre la mémoire physique et l'espace *swap* est géré par le *Page Frame Reclaiming Algorithm* (PRFA). Les deux opérations qui peuvent être effectuées sur l'espace *swap* par cet algorithme sont :

1. **Le *swap-out* :** qui consiste à écrire une page mémoire, présente dans la mémoire physique, dans l'espace *swap*. Elle se déroule selon les étapes suivantes :
   (a) Insertion de la page dans le cache de *swap* : l'algorithme vérifie que la page n'est pas déjà présente sur le cache [8] de *swap*. Si c'est le cas, l'algorithme alloue une page dans l'espace *swap* et écrit la page sur le cache.
   (b) Mise à jour de la table des entrées des pages : la table des entrées des processus associés à la page écrite dans le *swap* est mise à jour avec l'identifiant de la page dans l'espace *swap*.
   (c) Écriture effective de la page dans le *swap* : l'algorithme vérifie qu'il y a au moins un processus qui référence cette page. Si ce n'est pas le cas, la page est retirée du cache. Sinon, la requête d'écriture à l'adresse qui a été allouée dans l'espace *swap* est soumise à l'ordonnanceur d'E/S du disque. Une fois l'écriture terminée, les processus en attente de cette opération sont réveillés.
   (d) Retrait de la page du cache de *swap* : si aucun processus n'a essayé d'accéder à la page pendant l'opération de *swap-out*, l'espace occupé par la page sur le cache est libéré.
2. **Le *swap-in* :** cette opération consiste à copier une page de l'espace *swap* à la mémoire principale. Elle se déroule selon les étapes suivantes :
   (a) Récupération de l'adresse de la page dans l'espace *swap*.
   (b) Tentative de récupération de la page du cache de *swap* : si la page est dans le cache, aller directement à l'étape (d).
   (c) Lecture de la page du *swap* : un groupe de $2 \cdot n$ pages qui inclut la page demandée est lu de l'espace *swap* vers le cache de *swap*. n vaut 3 par défaut.
   (d) Vérification que la restitution n'a pas été déclenchée par un autre processus auquel cas la restitution n'est pas nécessaire.
   (e) Vérification du taux de remplissage du cache *swap*. S'il est supérieur à 50%, et que la page appartient à l'espace d'adressage d'un seul processus, la page est retirée du cache à l'issue de la restitution.
   (f) Mise à jour de la table des entrées du processus.

8. La cache de *swap* est un mécanisme introduit pour prévenir les problèmes de synchronisation des opérations de *swap* sur des pages partagées entre plusieurs processus.

Application	Architecture *Edge*	Algorithme de ML	Approche	Référence
Reconnaissance d'images	Décentralisée	DNN	Conception d'un modèle qui prend en considération les ressources disponibles sur la plateforme pour réaliser un compromis précision/taille du modèle.	[48]
N/A	Décentralisée	DNN	Conception d'une topologie mémoire adaptée au motif d'accès des DNN.	[146]
N/A	Décentralisée	*Transfert Learning*[9]/CNN	Allocation mémoire à la demande afin de limiter la mémoire occupée par l'application d'apprentissage.	[143]
N/A	Décentralisée	DNN	Exploitation de la mémoire partagée entre CPU et GPU d'un *smartphone* pour limiter les mouvements de données.	[144]
Reconnaissance d'images	Hybride	CNN	Utilisation d'approximations des poids du CNN afin de réduire l'empreinte mémoire. Formalisation en programmation linéaire de sorte à maximiser la précision du modèle sous les contraintes d'énergie, empreinte mémoire, etc.	[63]
Reconnaissance d'images	Décentralisée	CNN	Conception d'un accélérateur de calcul de convolutions économe en mouvements de données.	[28, 29, 152, 85]
N/A	N/A	CNN	Automatisation de la conception d'accélérateurs matériels de CNNs sous différentes contraintes, *Neural Architecture Search*.	[10]
Détection de mouvements	Décentralisé	Réseaux Baysiens	Apprentissage sur un sous-ensemble de données et non sur le *data-set* complet.	[78]
N/A	N/A	Random Forests	Maintien d'une proportion de données en mémoire entre la construction de deux arbres de décision consécutifs afin de limiter les mouvements de données.	[64]
N/A	Décentralisée	Arbre de décision	Proposition d'une nouvelle forme d'arbre de décisions appelée *Bonsai Tree* qui est plus compacte qu'un arbre de décision classique.	[76]
N/A	Décentralisée	*K*-NN	Projection des données du *data-set* sur un espace de dimension inférieure à la dimension de départ. Apprentissage sur des sous-ensembles de données.	[59]
N/A	N/A	N/A	Calcul *in-storage*.	[60, 87]
N/A	N/A	N/A	Calcul *in-memory*.	[9]

TABLE 2.5 – Synthèse des travaux sur la réduction de l'empreinte mémoire dans l'EI

9. Technique qui consiste à utiliser comme base les propriétés apprises à partir d'un CNN pour en entraîner un autre [143].

2.4.2 Solutions de réduction de l'empreinte mémoire pour l'*Edge* Intelligence

Dans cette section, nous synthétisons les méthodes de réduction de l'empreinte mémoire dans l'EI. Le tableau 2.5 résume les approches des travaux que nous allons décrire ci-après. Les travaux sélectionnés concernent uniquement l'opération d'entraînement des modèles d'apprentissage.

Solutions matérielles :

Le calcul *in-storage* : le calcul *in-storage* consiste à introduire des éléments matériels qui permettent de traiter les données sur les plateformes de stockage. L'avantage de cette approche est de réduire le temps nécessaire aux mouvements de données entre les niveaux de la hiérarchie mémoire en exportant le calcul vers le stockage [60]. Plusieurs travaux proposent alors de déléguer les calculs aux contrôleurs de SSD [75, 38]. Néanmoins, selon [124], cette approche présente plusieurs limitations : les faibles capacités de calculs des contrôleurs de SSD, l'absence d'une interface généraliste qui permet aux programmeurs de déléguer facilement le calcul au contrôleur de SSD, le manque de support système qui permet de gérer les cas de concurrences des processus, d'isolation et protection des données. Une autre approche consiste à utiliser un processeur dédié (ASIC, *Application-Specific Integrated Circuit*), pour concevoir un contrôleur de SSD spécifique à un application, par exemple : les bases de données [69], le *Big Data* [56] et les algorithmes de ML [60, 87].

Le calcul *in-memory/near-memory* : il consiste à exporter les calculs vers la mémoire principale [91]. Deux approches peuvent être distinguées : la réalisation des opérations arithmétiques des algorithmes de ML en utilisant la structure de la mémoire elle-même [91]. Dans [80], les auteurs proposent d'activer deux lignes mémoire (à base technologie NVM) pour implémenter les opérations logiques (AND, OR, XOR et INV) sans transférer les données vers le processeur. Dans [9], 19 travaux appliquant cette approche sont répertoriés, tous s'intéressent aux CNN/DNN. Parmi ces travaux, uniquement 3 s'intéressent à la phase d'apprentissage des modèles de CNN/DNN : [79] sur la DRAM, [135] sur la ReRAM et [6] sur la SOT-MRAM. La deuxième approche consiste à introduire des éléments matériels de calcul tels que des accélérateurs ou des processeurs, près de la mémoire, par exemple, au niveau du contrôleur de mémoire ou au niveau des puces mémoire.

Dans [143], les auteurs proposent SCALEDEEP, un *framework* d'entraînement de réseaux de neurones qui génère une topologie matérielle adaptée au motif d'accès des

réseaux de neurones. L'architecture générée contient deux composants de base : un composant d'accélération de calculs, et un composant d'interconnexion (mémoire) des composants de calculs qui minimise les accès à la mémoire extérieure à l'accélérateur.

Eyeriss [28] s'inscrit dans la même optique de minimisation des mouvements de données entre l'accélérateur de calcul de convolutions et de la mémoire extérieure. Pour cela les auteurs proposent l'agencement des éléments de calculs de sorte à réutiliser au maximum les vecteurs des filtres de convolutions lorsque ils sont chargés dans l'accélérateur, ainsi que deux niveaux mémoire, un *scratch-pad* local à chaque unité de calcul et un buffer global partagé par toutes les unités de calcul. Eyeriss V2 [29] reprend l'architecture d'Eyeriss et y ajoute un réseau sur puce (NoC) qui relie le buffer global aux unités de calcul à moindre coût.

COSY [152] se base également sur l'architecture d'Eyeriss et se distingue par rapport à la dernière en la transformant en réseau systolique afin de rendre les *scratch-pad* locaux d'une unité de calcul partagée avec les unités de calcul qui lui sont adjacentes.

Plusieurs travaux ont proposé des accélérateurs matériels de CNN/DNN pour FPGA économes en ressources [159, 109, 85]. Les travaux [30, 40] s'intéressent exclusivement aux applications de reconnaissance d'images. Ils proposent des accélérateurs de CNN sur ASIC.

Au vu de la complexité de conception des accélérateurs de CNNs, due à la complexité inhérente aux réseaux de neurones d'une part et à l'hétérogénéité des plateformes matérielles et des contraintes auxquelles elles sont soumises d'autres part, plusieurs travaux, répertoriés dans [10], visent à automatiser la conception des CNNs en tenant compte de l'architecture matérielle et des contraintes à satisfaire. Cette technique est appelée *Neural Architecture Search* (NAS).

Solutions applicatives :

Dans [48], les auteurs proposent de concevoir un modèle de réseau de neurones qui s'adapte aux ressources disponibles sur la plateforme sur laquelle il s'exécute. Pour ce faire, plusieurs modèles sont générés hors ligne sous différentes contraintes de ressources disponibles, puis rassemblés en un seul modèle. Au moment du déploiement, le modèle choisi dépend des ressources effectivement disponibles.

Dans [63] les auteurs se basent sur le principe d'approximation des matrices de poids dans les réseaux de neurones, qui a pour objectif de réduire la taille du modèle en acceptant une perte de précision. L'approche proposée est de prendre en considération les ressources disponibles sur la plateforme embarquée afin de déterminer quelles approximations utiliser pour réaliser un bon compromis précision/ressources. Pour ce faire, les

auteurs formulent le problème en programmation linéaire où l'objectif est de maximiser la précision, tout en respectant les contraintes en ressources.

Au vu de la démocratisation de l'utilisation des CNN/DNN sur les plateformes embarquées, plusieurs *frameworks* visant à permettre l'apprentissage sous des contraintes en ressources soulevée par l'embarqué. TensorFlow Lite [139] a été proposé par *Google*, suivi de Caffe2 [21] et PyTorch [108] par *Facebook*, et de MXNet [27] par *Amazon*.

Discussion :

Dans cette section, nous avons passé en revue quelques travaux et approches qui visent à réduire l'empreinte mémoire et les mouvements de données entre les niveaux de la hiérarchie mémoire lors de l'entraînement d'algorithmes de ML pour l'EI. L'observation que l'on en tire est que la majeure partie des travaux sur l'EI est orientée vers les réseaux de neurones et les réseaux de neurones convolutionnels. Or, d'autres algorithmes d'apprentissage conviennent mieux à certaines tâches qui doivent être accomplies sur les dispositifs *edge*, par exemple, la détection d'intrusions où les RF sont largement utilisés. Par conséquent, il est intéressant d'analyser le motif des accès mémoire et des E/S et de les réduire.

2.4.3 Solutions d'optimisation du *K-means*

Au vu du peu de travaux menés pour la réduction des mouvements de données dans le cas de l'algorithme *K-means* dans l'EI, nous avons élargi le champs des travaux qui seront exposés dans cette section au-delà de l'EI. En effet, la problématique de l'empreinte mémoire et mouvements de données du *K-means* a été abordée notamment dans le contexte du *Big Data*. Dans ce qui suit, nous passons en revue les principales méthodes de *clustering K-means* dont l'objectif est de réduire son empreinte mémoire.

Réduction des calculs

L'une des méthodes de réduction des calculs qui conservent les résultats du *K-means* est donnée dans [44]. Cette méthode consiste à élaguer des calculs de distances à chaque itération du *K-means* en se basant sur les propriétés géométriques suivantes : soit x une observation, c et c' des centres au début d'une itération donnée :

— Si $d(x,c) \leq 1/2 \cdot d(c,c')$ alors $d(x,c) \leq d(x,c')$, le calcul de la distance $d(x,c')$ peut être évité ;

— La connaissance d'une borne supérieure u de $d(x,c)$ suffit à éviter le calcul de la distance $d(x,c')$ en vertu de la propriété précédente si $u \leq 1/2 \cdot d(c,c')$;

— Si $u \leq 1/2 \cdot min_{c' \neq c}\{d(c, c')\}$, alors x reste assigné à son centre actuel, les distances entre x et les centres restants n'ont pas besoin d'être calculés ;
— Soit b la version du centre c à l'itération précédente, tel que la borne inférieure de la distance $d(x, b)$ est donnée par l. On a la propriété $d(x, c) \geq max\{0, l - d(b, c)\}$;
— On suppose que x est assigné au centre c. Supposant que $d(x, c) \leq u$ et $d(x, c') \geq l$. Si $u \leq l$ alors $d(x, c) \leq u \leq c \leq d(x, c')$. On n'a, alors, pas besoin de calculer les distances $d(x, c)$ et $d(x, c')$

En somme, ces propriétés permettent d'éviter le calcul de plusieurs distances entre une observation x et les centres, si x vérifie certaines propriétés. Par ailleurs, cet élagage n'altère pas les résultats obtenus par rapport au *K-means* classique. Dans le contexte de la réduction des mouvements de données, éviter le calcul de cette distance revient à éviter d'accéder à l'observation x, et par suite, éviter de charger le bloc qui la contient. Néanmoins, cette méthode requiert tout de même le maintien en mémoire de nouvelles structures pour contenir les bornes inférieures et supérieures des distances entre les observations et un centre quelconque. D'autres travaux ont proposé des approches d'élagage de calculs de distances : [62, 39, 37, 13] proposent des améliorations en terme de coût des structures maintenues en mémoire par [44].

Parallélisation du *K-means*

Dans [35], les auteurs ont pour objectif d'exploiter le parallélisme des processeurs graphiques (GPU) pour accélérer l'algorithme du *K-means*. Les auteurs commencent par proposer une version naïve de la parallélisation de l'algorithme *K-means* sur GPU, qui est donné dans l'algorithme 3. L'approche consiste à distribuer, le calcul des distances des observations par rapport aux centres et l'affectation de l'observation au centre le plus proche, sur plusieurs *threads*. Puis, à la fin de chaque itération, l'un des *threads* (ici le 0) effectue le re-calcul des centres et la vérification de la convergence. Néanmoins, dans cette solution, si le volume des observations à traiter par un *thread* est plus grand que ce que peut contenir la mémoire d'un *thread* du GPU, les mouvements de données entre les niveaux de la hiérarchie mémoire du GPU subsistent. Les auteurs proposent alors de distribuer les données de la façon suivante :

— Chaque *thread* du GPU traite une composante de l'observation x. Un bloc est composé de d *threads*, ce qui fait que chaque bloc traite une observation ;
— Le calcul associé à une composante d'observation donnée consiste à calculer la distance euclidienne par rapport à un centre donnée. L'information contenue dans un *thread* i est la composante i de l'un des centres pour pouvoir calculer la distance qui les sépare ;

Algorithm 3: Algorithme de parallélisation du *K-means*

```
Data: D : Data-set
Result: K centres
if Numéro de thread = 0 then
    Initialiser de manière aléatoire les K centres
Synchroniser les threads
repeat
    foreach x ∈ observations affectées au thread i do
        Affecter x au centre qui lui est le plus proche
    Synchroniser les threads
    if Numéro de thread = 0 then
        Mettre à jour les K centres
        if Les centres ont convergé then
            Signaler aux autres threads que l'algorithme a convergé
until les centres ont convergé;
```

— Une fois toutes les distances de composantes calculées, leur somme est calculée au niveau bloc.

L'avantage de cette méthode est que dans le cas où le nombre d'observations est bas, le nombre de mouvements de données est bas (le meilleur cas étant qu'il y ait autant de blocs dans le GPU que d'observations). Outre la réduction des mouvements de données, les distances entres les observations et les centres lors d'une itération sont calculées en parallèle. En revanche, dans le cas où le nombre d'observations est supérieur à ce que peut contenir le GPU, la problématique des mouvements de données entre la hiérarchie mémoire se pose de nouveau. D'autres travaux antérieurs à [35] ont pour objectif la parallélisation du *K-means* en exploitant la puissance du GPU [81].

Réduction du nombre d'itérations du *K-means*

Dans [101], les auteurs proposent une méthode adaptée à un gros volume de données et avec une dimension de problème d élevée. Pour ce faire, la solution repose sur une première optimisation algorithmique, dont l'objectif est de réduire le nombre de parcours sur le *data-set*, et une seconde méthode, dont le but est d'organiser les résultats sur le disque.

La première optimisation consiste à mettre à jour les centres $\vec{\mu}_1, \vec{\mu}_2, ..., \vec{\mu}_K$ avec une périodicité de $\sqrt{N}$, où N est le nombre d'observations du *data-set*. Autrement dit, au lieu de recalculer les centres à la fin de chaque itération comme dans le *K-means* classique, la mise à jour des centres se fait plus fréquemment, à raison de $\sqrt{N}$ calculs de distances. Cela permet de réduire le nombre de parcours sur le *data-set* et donc le volume des E/S.

Les auteurs, fixent de manière arbitraire, le nombre de parcours à 3.

L'optimisation en terme d'organisation des données sur le disque consiste à stocker les données liées au *clustering* sous forme de tables de la manière suivante :

— La table du *data-set* : le *data-set* est stocké dans une table dont les attributs sont : l'identifiant de l'observation dans le *data-set*, l'identifiant de la propriété [10], et la valeur de la propriété pour l'observation en question. Lorsque la propriété d'une observation vaut 0, elle n'est pas enregistrée dans la table, afin de stocker le *data-set* de manière compact. L'intérêt de stocker dans la table, l'identifiant de la propriété est la réduction de l'espace occupé par le *data-set* dans le cas où il est éparse (*sparse matrix*). Ainsi, pour un problème de dimension d, une observation $\vec{x}_i$ est stockée sur au plus d lignes de la table.

— La table d'affectation des observations : cette table contient les couples (identifiant de l'observation, *cluster* auquel elle est affectée).

— Table des centres : dans cette table, chaque ligne contient l'identifiant du centre, l'identifiant de la propriété et la valeur de la propriété du centre du *cluster*. La table des variances est structurée de la même manière que la table des centres.

Méthodes de *K-means* basées sur le principe de « diviser pour régner »

Plusieurs travaux ont été basés sur le principe d'application du *K-means* sur des sous-ensembles du *data-set* puis de combinaison des résultats.

Algorithme naïf : cette méthode est donnée comme méthode de base dans [57]. Elle est décrite dans l'algorithme 4. Elle consiste à diviser le *data-set* en l sous-ensembles disjoints $\{S_1, S_2, ..., S_l\}$ (ligne 1 de l'algorithme 4). Pour chaque sous-ensemble S_j, le *K-means* est appliqué (lignes 3-7). Le *K-means* est appliqué sur les $l \cdot K$ centres pondérés par le nombre d'observations qui leurs sont affectées (ligne 8). Cet algorithme ne prend pas en compte la distribution des observations au sein d'un *cluster*. Cela est dû au fait de prendre en compte uniquement les centres obtenus des l *K-means* pour représenter les *clusters*. Ainsi, si les *clusterings* obtenus pour les sous-ensembles ont des erreurs importantes (la variance intra-*clusters* est importante), le fait de représenter chaque *cluster* par le point central ne permet pas de rectifier l'erreur dans la phase du *clustering* final.

Compromis entre la rapidité de calcul et la qualité du *clustering* : afin d'améliorer la qualité du *clustering* par rapport à l'algorithme naïf, les auteurs proposent dans [57]

10. Soit $\vec{x}$ un tuple qui représente une observation, la $i^{ème}$ composante de ce tuple est identifiée par i.

Algorithm 4: Algorithme Simple du *K-means* selon le principe diviser pour régner

Data: D : Data-set
Result: K centres
Diviser D en l sous-ensembles disjoints $\{S_1, S_2, ..., S_l\}$
for $j \leftarrow 0$ **to** $l - 1$ **do**
    Initialiser les K centres pour le sous-ensemble S_j
    while *Non convergence* **do**
        for $i \leftarrow 0$ **to** $|S_j| - 1$ **do**
            Affecter l'observation i du data-set au centre le plus proche
        Recalculer les K centres pour le sous-ensemble S_j
Appliquer le *K-means* sur les centres pondérés obtenus pour les l sous-ensembles

d'introduire une constante a dans l'algorithme 4 pour former $a \cdot K$ centres au lieu de K. L'objectif est d'avoir plus de points représentatifs de chaque sous-ensemble S_j. Le nouvel algorithme est décrit dans l'algorithme 5.

Algorithm 5: Algorithme Simple du *K*-means selon le principe diviser pour régner

Data: D : Data-set, a : constante
Result: K centres
Diviser D en l sous-ensembles disjoints $\{S_1, S_2, ..., S_l\}$
for $j \leftarrow 0$ **to** $l - 1$ **do**
    Initialiser les $a \cdot K$ centres pour le sous-ensemble S_j
    while *Non convergence* **do**
        for $i \leftarrow 0$ **to** $|S_j| - 1$ **do**
            Affecter l'observation i du data-set au centre le plus proche
        Recalculer les $a \cdot K$ centres pour le sous-ensemble S_j
Appliquer le *K-means* sur les centres pondérés obtenus pour les l sous-ensembles

La constante a permet de réaliser un compromis entre le nombre d'itérations nécessaires au *clustering* des sous-ensembles S_j et la qualité du *clustering*. En effet, plus le nombre de centres est grand, plus le nombre d'itérations nécessaires à la convergence est grand ce qui ralentit l'algorithme. Le fait d'augmenter le nombre de centres pour chaque sous-ensemble permet quant à lui d'augmenter le nombre d'observations représentatives pour la phase de *clustering* finale, ce qui améliore la qualité du *clustering*. Le principal problème de cet algorithme est qu'il est difficile de fixer la constante a de manière à réaliser un compromis satisfaisant.

Diviser pour régner, une approche collaborative : les auteurs proposent dans [34] une méthode pour fixer la valeur de la constante a basée sur ces deux principes :

- Le *clustering* collaboratif des sous-ensembles du *data-set* : le *K-means* est appliqué sur les sous-ensembles du *data-set* de manière séquentielle. L'objectif est d'utiliser les centres obtenus pour le sous-ensemble j comme centres initiaux pour le sous-ensemble $j+1$. De cette façon, si la distribution des observations sur les sous-ensembles est uniforme, cette initialisation des centres permettrait de réduire le nombre d'itérations nécessaires à la convergence.
- La méthode de combinaison des résultats : afin d'améliorer la qualité du *clustering*, les auteurs proposent une méthode de rectification des résultats. Pour cela, ils ajoutent une phase qui s'effectue après l'étape de combinaison des résultats du *clustering* (ligne 8 de l'algorithme 4), qui consiste à ré-appliquer le *K-means* sur les sous-ensembles qui sont à cheval entre deux *clusters* globaux. Soient $G_1, G_2, ..., G_K$ les centres des *clusters* globaux, pour chaque centre partiel $\vec{\mu}_i$ l'ensemble $NS(\vec{\mu}_i, G)$ des centres globaux qui sont les plus proches à $\vec{\mu}_i$ est calculé selon la définition qui sera donnée dans la définition 12. Si le nombre des centres qui appartiennent à cet ensemble est supérieur à 1, le *K-means* est ré-appliqué aux observations dont les sous-ensembles appartiennent aux *clusters* globaux de l'ensemble $NS(\vec{\mu}_i, G)$.

Définition 12. *L'ensemble $NS(\vec{\mu}_i, G)$ des clusters globaux les plus proches à $\vec{\mu}_i$ est défini comme suit : Soit G l'ensemble des centres globaux. $\eta(\vec{\mu}_i, G)$ le centre global le plus proche à $\vec{\mu}_i$.*

$$NS(\vec{\mu}_i, G) = \{G_j | \Omega(\vec{\mu}_i, G_j) < (1+\epsilon) \cdot \eta(\vec{\mu}_i, G)\} \tag{2.8}$$

Où $\Omega(\vec{\mu}_i, \vec{\mu}_j)$ est la distance pondérée entre deux centres et est donnée par l'équation :

$$\Omega(\vec{\mu}_i, \vec{\mu}_j) = \frac{|C_i| \cdot |C_j|}{|C_i| + |C_j|} |\vec{\mu}_i - \vec{\mu}_j|^2 \tag{2.9}$$

C_i et C_j sont les clusters associés aux centres $\vec{\mu}_i$ et $\vec{\mu}_j$.

Mini-Batch *K-means*

Dans [128], les auteurs proposent d'effectuer le *clustering* sur t petits sous-ensembles (*batch*) du *data-set* dont les observations sont échantillonnées aléatoirement et de mettre à jour les centres des *clusters* après traitement de chaque *batch*. L'algorithme est décrit dans l'algorithme 6.

La méthode de descente de gradient a été proposée dans [14], elle a pour but d'accélérer la convergence de l'algorithme. Son principe est de mettre à jour les centres

Algorithm 6: Algorithme du mini-batch K-means

Data: D : Data-set, b : taille du *batch*, t : nombre d'itérations
Result: K centres
Initialiser les centres de manière aléatoire
for $i \leftarrow 1$ **to** t **do**
    $M \leftarrow b$ observations tirées aléatoirement de D
    for $x \in M$ **do**
        Affecter l'observation x au centre le plus proche
    for $x \in M$ **do**
        Mettre à jour le centre auquel x est affecté selon la méthode de descente de gradient

non pas en calculant la moyenne des observations affectées au centre (équation (2.2)), mais en se basant sur une méthode analytique qui consiste à trouver une valeur approchée du centre qui annule le gradient de l'erreur [7]. Soit f la fonction d'erreur de *clustering* (par exemple, la variance intra-*cluster* donnée dans la définition 2.1 page 42), ϵ l'erreur tolérée et α un pas tel que $\epsilon > 0$ et $\alpha > 0$. L'algorithme du gradient est donné dans l'algorithme 7. Dans l'algorithme de descente de gradient, il est important de bien définir le paramètre

Algorithm 7: Algorithme de la descente de gradient

Data: f : la fonction d'erreur, ϵ : l'erreur tolérée, α : le pas
Result: $\vec{\mu}$: Centre approximatif qui minimise l'erreur
Choisir un μ initial aléatoirement
while $\|\nabla f(\vec{\mu})\|_2^2 > \epsilon$ **do**
    $\vec{\mu} \leftarrow \vec{\mu} - \alpha \nabla f(\vec{\mu})$

α, car si α est très faible, le nombre d'itérations nécessaires à la convergence sera élevé. À l'inverse, si α est très grand l'algorithme peut diverger si le centre qui minimise l'erreur est dépassé [7].

Discussion

Dans cette partie, nous avons passé en revue quelques approches de l'état de l'art sur les méthodes de *K-means* économes en mémoire. Globalement, il en ressort les principes suivants :

1. L'élagage des calculs de distance qui permet de ne pas d'accéder certaines observations [44].
2. Diviser pour régner : cette stratégie consiste à appliquer le *K-means* sur des sous-ensembles du *data-set* puis d'agréger les résultats. À travers cet état de l'art, nous

remarquons qu'il existe plusieurs approches dans la façon de former les sous-ensembles. Dans [57, 34] les auteurs forment des sous-ensembles disjoints dont l'union donne le *data-set* entier. En revanche, dans [128] les sous-ensembles sont échantillonnés de manière aléatoire.

3. Le re-calcul anticipé des centres : dans [101], il est fait avec une périodicité de $\sqrt{N}$; dans [34] cela est fait à la fin du traitement de chaque sous-ensemble du *data-set*. Dans ces deux méthodes, cela est fait en utilisant la méthode de mise à jour classique des centres comme étant les barycentres des *clusters*. Dans [128], la mise à jour se fait à chaque itération et utilise la méthode de descente de gradient pour accélérer la convergence vers les centres optimaux.
4. La distribution des données sur la hiérarchie mémoire disponible sur le matériel pour limiter les mouvements de données [81, 35].

Les principales limitations des méthodes discutées dans cette synthèse de l'état de l'art sont les suivants :

1. Elles affectent la qualité des résultats obtenus : les méthodes [34], [57] retournent des résultats moins précis que la méthode de *K-means* traditionnelle. Dans [128], pour obtenir des résultats précis il faut augmenter le nombre d'itérations t. Néanmoins, cette méthode est considérée comme efficace et a été mise en œuvre dans Sci-Kit Learn [105].
2. Les méthodes basées sur le principe « diviser pour régner » ne prennent pas en compte le volume de mémoire physique disponible sur la plateforme : ces méthodes ne définissent pas explicitement la taille des sous-ensembles qu'il serait intéressant de fixer pour réaliser un bon compromis précision/temps.

2.4.4 Solutions d'optimisation des Random Forests (RF)

Dans cette dernière section du chapitre, nous nous intéressons aux travaux menés pour réduire les mouvements de données pour l'algorithme de construction des RF. Comme pour le *K-means*, nous avons élargi le spectre de recherche au-delà de l'EI au vu du peu de travaux menés sur les RF.

La méthode de construction hybride en profondeur et en largeur

Dans [5], les auteurs évaluent et identifient les goulets d'étranglements des constructions en largeur et en profondeur des arbres de décision en terme d'accès à la hiérarchie

mémoire. Cette étude leur a permis de proposer une méthode hybride de construction en largeur puis en profondeur des arbres de décision.

Construction en profondeur, ou *Depth-First-Search* (DFS) : dans cette construction, après qu'un nœud ait été divisé, le nœud suivant à traiter est son nœud enfant gauche. Ce dernier est traité entièrement avant de passer au nœud droit.

Construction en largeur, ou *Breadth-First-Search* (BFS) : ici, les nœuds sont traités par niveau de profondeur. Autrement dit, après traitement d'un nœud donné, le nœud suivant à être traité est le nœud se situant immédiatement à sa droite sur le même niveau. S'il n'existe pas, le traitement est effectué sur le nœud le plus à gauche sur le niveau suivant.

Dans le travail [5], les auteurs analysent le motif d'accès à la mémoire des deux méthodes de construction. Il est montré que la construction BFS est efficace sur les premiers niveaux de l'arbre puisque les observations du *bootstrap* sont distribuées sur peu de nœuds, ce qui fait que lorsqu'un bloc est accédé et que le bloc contient des observations qui ne sont pas affectées au nœud courant, il est probable que ces observations appartiennent à un nœud se trouvant sur le même niveau. Par conséquent, cette méthode de construction favorise la localité spatiale. En revanche, lorsque la profondeur de l'arbre augmente, le nombre d'observations actives (qui appartiennent à des nœuds qui doivent encore être divisés) se réduit. L'algorithme ne profite alors plus de la localité comme dans les premiers niveaux. À l'inverse, la performance de la construction DFS s'améliore avec la profondeur de l'arbre. En effet, traiter le nœud enfant immédiatement à gauche permet de réutiliser les observations du nœud courant dans le nœud fils gauche, puisque les nœuds contiennent de moins en moins d'observations à mesure que la pronfondeur de l'arbre augmente.

Méthode de construction hybride BFS-DFS. Les auteurs [5] proposent une méthode de construction hybride qui consiste à commencer par une construction en largeur sur les premiers niveaux de l'arbre. À chaque niveau de l'arbre, le nombre d'observations encore actives (qui servent à la division des nœuds impures) est monitoré. Lorsque ce nombre est trop petit, l'algorithme de construction ne bénéficie plus de la localité inhérente à la construction en largeur, la construction en profondeur est donc utilisée. Le nombre d'observations actives utilisé pour le passage de la construction BFS à une construction DFS est atteint lorsque les observations actives peuvent être contenues dans le cache du CPU.

Le *framework* Ranger

Dans [149], les auteurs proposent une mise œuvre rapide des RF, basée sur l'optimisation de l'utilisation de la mémoire. L'optimisation la plus importante dans ce *framework* est effectuée lors la phase de la division du nœud (voir les lignes 4-5 de l'algorithme 2). Les auteurs proposent deux algorithmes pour effectuer la division :

1. Dans le premier, l'objectif est de faire de l'élagage, c'est à dire, éviter d'accéder à certaines observations. Pour ce faire, toutes les valeurs des propriétés des observations sont triées, après chargement du *data-set* en mémoire. Un index permettant de retrouver la valeur associée à la propriété de chaque observation est construit. Le tri des valeurs des propriétés permet d'accélérer l'opération de division. En effet, pour un nœud donné, si la valeur de division est définie à v, lorsque la première valeur de propriété supérieure à v est rencontrée, le reste des observations affectées au nœud en cours de division sont affectées au nœud enfant droit (les autres sont affectées au nœud enfant gauche). Cela permet donc de parcourir moins de valeurs de propriétés. Le tri se fait au lancement de la construction de la RF et l'index est maintenu tout au long de la construction.
2. Le deuxième algorithme consiste à trier les valeurs de propriétés à la demande. C'est à dire que seules les valeurs des propriétés nécessaires à la division d'un nœud sont triées, et le tri se fait immédiatement avant la division du nœud.

D'un point de vue compromis rapidité de calcul-usage de la mémoire, le premier algorithme accélère la division mais l'index nécessaire est volumineux. Le deuxième algorithme est plus économe en mémoire puisque les index à construire à chaque nœud ne concernent que les propriétés à tester ($\sqrt{d}$ pour les arbres de décision destinés à la classification) et concerne de moins en moins d'observations à mesure que la profondeur de l'arbre augmente. En revanche, dans cette deuxième version, le tri est lancé à chaque division de nœud. Par conséquent, les auteurs optent pour l'utilisation du premier algorithme pour les nœuds dont le nombre d'éléments est grand pour accélérer la division, et le deuxième algorithme pour les nœuds qui contiennent peu d'éléments (le tri étant plus rapide et le parcours de toutes les observations du nœud moins coûteux). De plus, les auteurs intègrent dans *Ranger* un mode *économie en mémoire* (*Memory-Efficient*) dans lequel seul le deuxième algorithme est utilisé.

Méthode des RF adaptée aux NVM

Dans [64], les auteurs proposent une méthode de construction des RF adaptée aux plateformes embarquées dont la mémoire principale est limitée, ce qui engendre des mouve-

ments de données vers la mémoire secondaire (*swap*). La mémoire secondaire étant basée sur la technologie NVM, son endurance est limitée et le temps d'écriture est long comparé à la mémoire principale. La méthode proposée vise à réduire les mouvements de données entre la mémoire principale et l'espace de *swap* lors de la construction des arbres de décision de la RF.

Pour cela, les auteurs se basent sur le fait que la construction d'un arbre de décision se fait sur un échantillon aléatoire (*bootstrap*) des observations du *data-set* de départ. Par conséquent, pour la construction de deux arbres consécutifs, une proportion des observations comprises dans le premier *bootstrap* peut être échantillonnée pour le *bootstrap* du deuxième arbre de décision. Théoriquement, une observation a une probabilité de 0.632 d'être échantillonnée dans un *bootstrap* [19]. Les auteurs se basent sur cette propriété et proposent d'inclure, a priori, une proportion des observations utilisées pour un *bootstrap* i dans le *bootstrap* $i+1$. De cette manière, ces observations peuvent être gardées en mémoire, et réduire ainsi le volume des E/S nécessaires au chargement des données de l'arbre suivant. Dans cette étude, la proportion des observations à garder pour le *bootstrap* suivant est variée de façon à déterminer, par l'expérience, la proportion qui n'affecte pas la qualité des résultats obtenus. L'expérience montre qu'au delà de 75% d'observations réutilisées, la qualité des résultats se dégrade.

La deuxième contribution de cette étude est une méthode qui prend en compte le nombre d'écritures que supporte la NVM dans le choix des observations à garder en mémoire (réutiliser pour le *bootstrap* suivant). Pour cela, les auteurs proposent d'utiliser un pivot qui sert à délimiter la zone dans laquelle les observations sont réutilisées. Ainsi, les observations à réutiliser pour le *bootstrap* $i+1$ sont celles qui se trouvent entre la position courante du pivot et le nombre d'observations nécessaires pour atteindre la proportion fixée d'observations à réutiliser. Une fois le traitement de ce *bootstrap* fini, le pivot est avancé jusqu'à la position à partir de laquelle aucune observation n'a encore été réutilisée. À terme, cette méthode permet d'équilibrer le taux d'écriture sur les blocs de la NVM.

Discussion

À travers cette synthèse de quelques travaux qui ont été menés pour réduire l'empreinte mémoire pour les RF, nous relevons trois idées principales :

1. Le parcours en profondeur DFS ou en largeur BFS ont une influence sur la localité des données. C'est ce qui est montré dans [5].
2. [149] montre que le tri des propriétés des observations permet de réduire le nombre

d'observations parcourues mais augmente l'empreinte mémoire.

3. Dans [64], les auteurs exploitent le fait qu'une proportion des observations du *data-set* sont échantillonnées pour des *bootstraps* consécutifs. Par conséquent, ils proposent « d'épingler » une proportion des observations en mémoire entre la construction de deux arbres consécutifs pour ne pas avoir à les recharger.

Néanmoins, ces méthodes gardent l'hypothèse que le *data-set* est chargé avant le début de la construction de la RF pour [5, 148], et avant le début de construction d'un arbre de décision pour [64]. Or, les E/S dus au mécanisme du *swap* résultent de cette hypothèse.

2.5 Conclusion

Dans ce chapitre, nous avons passé en revue le paysage de l'EI, à savoir ses architectures possibles, ses composantes, et la taxonomie des travaux de recherche s'y rapportant. Les trois composantes principales de l'EI sont : les données, les algorithmes de ML et le calcul. Nous avons, donc, décrit les principaux algorithmes, puis en avons choisi deux comme cas d'étude pour cette thèse, à savoir : le *K-means* et les Random Forests. Notre attention s'est portée sur ces deux algorithmes au vu de leur popularité et leur précision.

L'EI soulève plusieurs défis que nous avons décrits par la suite. Dans cette thèse, nous nous focalisons sur le défis de réduction de l'empreinte mémoire pour l'EI. En effet, les algorithmes de ML doivent être exécutés sur de gros volumes de données, alors que les ressources disponibles sur les dispositifs embarqués sont limitées. Plus concrètement, la problématique à laquelle nous voulons répondre dans cette thèse est l'analyse et la réduction des E/S engendrées par les mouvements de données entre la mémoire physique et l'espace de *swap* pendant l'exécution des algorithmes de ML, lorsque le volume des *data-set* à traiter dépasse le volume de mémoire de travail disponible. Afin de mieux comprendre cette problématique, nous avons décrit la hiérarchie mémoire, et les conséquences d'une mémoire de travail limitée par rapport aux besoins. Par la suite, nous avons passé en revue les méthodes de réduction des mouvements de données dans l'EI. Nous notons que la majorité des travaux s'intéressent aux DNN/CNNs. Par ailleurs, les travaux qui concernent nos cas d'étude, supposent que le *data-set* réside en mémoire. Or, les mouvements de données sont causés par cette hypothèse. Par conséquent, nos contributions s'articulent autour de l'élimination de cette hypothèse puisque le volume de données amené à être traité pour l'EI augmente de manière exponentielle.

Chapitre 3

Algorithme *K-means* économe en E/S, K-MLIO

Introduction

Dans ce chapitre, nous présentons notre première contribution visant la réduction des E/S pour les algorithmes de ML dans le contexte des systèmes embarqués. Le premier cas d'étude auquel on s'intéresse dans cette thèse est l'algorithme de *clustering K-means*. Dans notre synthèse bibliographique, nous avons montré que plusieurs optimisations permettent de réduire l'empreinte mémoire de cet algorithme. Néanmoins, aucun de ces travaux n'a souligné le problème des mouvements de données du *data-set* entre la mémoire physique et l'espace de stockage secondaire, spécifiquement l'espace *swap*. Nous nous intéressons, alors, à la réduction des E/S dus à ces mouvements afin d'accélérer l'algorithme du *K-means*. Pour cela, nous commençons par caractériser, théoriquement et expérimentalement, le comportement de l'algorithme du point de vue des mouvements de données entre la mémoire principale et le *swap*. Ensuite, nous proposons une révision de l'algorithme, de sorte qu'il soit plus économe en E/S. L'algorithme proposé est soumis à une évaluation. Nous concluons le chapitre par une discussion sur l'apport de la proposition.

Sommaire

3.1 Analyse des accès mémoire l'algorithme *K-Means*

Notre objectif dans cette section est de caractériser les accès au stockage secondaire par l'algorithme *K-means* en identifiant la cause de ces accès et en les évaluant quantitativement. Nous effectuons, en premier lieu, une analyse et évaluation théoriques du volume des E/S qui surviennent, étant donné le volume du *data-set* à traiter et l'espace mémoire disponible. En second lieu, nous évaluons, de manière expérimentale, l'effet des E/S sur le temps d'exécution de l'algorithme lorsque la volume du *data-set* est supérieur à l'espace mémoire disponible.

3.1.1 Analyse théorique des E/S de l'algorithme *K-means*

Dans ce qui suit, nous ignorons volontairement le niveau de cache processeur. En effet, au vu de l'hypothèse que le volume du *data-set* dépasse l'espace mémoire disponible et étant donné le temps d'accès des E/S qui résultent de cette hypothèse, l'effet du cache est négligeable. On suppose que l'architecture est constituée d'une mémoire principale, et d'un espace de stockage secondaire dont une partie représente l'espace de *swap*. Dans cette section, nous souhaitons analyser l'évolution du volume de données échangées entre l'espace mémoire et l'espace de *swap* durant la phase d'apprentissage par l'algorithme *K-means*. Cette analyse permet d'identifier l'origine algorithmique de ces accès au stockage secondaire.

On pose N comme le nombre d'observations contenues dans le *data-set*. On suppose que M est le nombre d'observations qui peuvent être contenues dans l'espace mémoire physique. La taille S qui peut être contenue en mémoire est alors $S = M \cdot s$, où s est la taille d'une observation.

Comme le montre l'algorithme du *K-means* donné dans l'algorithme 1 page 43, à chaque itération de l'algorithme, toutes les observations du *data-set* sont parcourues. Tra-

ditionnellement, le *data-set* est stocké sous-forme de matrice et est gardée en mémoire pendant toute la phase d'apprentissage. Sous l'hypothèse que l'espace mémoire peut contenir M observations et que $N > M$, le reste des observations du *data-set* sont contenues dans l'espace de *swap* et doivent être chargées en mémoire à chaque itération de l'algorithme classique.

On suppose qu'un bloc d'E/S peut contenir B observations. Le nombre de blocs contenus dans l'espace mémoire disponible est de $\lfloor M/B \rfloor$, et le parcours complet du *data-set* revient à charger $\lceil N/M \rceil$ portions (*chunks*) du *data-set* en mémoire. Par conséquent, le volume d'E/S causées par les mouvements de données entre la mémoire physique et l'espace *swap* à l'exécution d'une itération de l'algorithme *K-means* est :

$$I/O_{cost\_per\_iter} = \lceil N/M \rceil \cdot \lfloor M/B \rfloor \tag{3.1}$$

Dans l'équation (3.1), $\lceil N/M \rceil$ traduit le nombre de *chunks* (parties du *data-set* dont la taille est égale à l'espace mémoire disponible) dans le *data-set* et sera utilisé dans le reste de cette thèse pour symboliser la contrainte mémoire. $\lfloor M/B \rfloor$ quant à lui traduit le nombre de blocs pour un *chunk*.

Si l'algorithme du *K-means* nécessite I itérations pour converger, le coût total en E/S est alors :

$$I/O_{cost} = \lceil N/M \rceil \cdot \lfloor M/B \rfloor \cdot I \tag{3.2}$$

L'équation (3.2) montre que le coût en E/S entre la mémoire physique et l'espace de *swap* de la version originale du *K-means* dépend de deux paramètres liés à l'application :

— Le volume du *data-set* par rapport à l'espace mémoire disponible $\lceil N/M \rceil$;
— La vitesse de convergence de l'algorithme I : dans [145], les auteurs montrent que le nombre d'itérations nécessaires à la convergence de l'algorithme *K-means* a une borne supérieure dont la valeur est exponentielle en fonction de N, le nombre d'observations du *data-set*, et K le nombre de *clusters* à trouver. Le nombre d'itérations nécessaires à la convergence de l'algorithme peut donc augmenter rapidement lorsque le volume de données à traiter augmente.

Une manière de baisser le coût des E/S de cet algorithme est de décorréler le volume des E/S (équation (3.2)) de la vitesse de convergence I. La méthode que nous proposons repose sur cette idée et sera décrite dans la section 3.2.1.

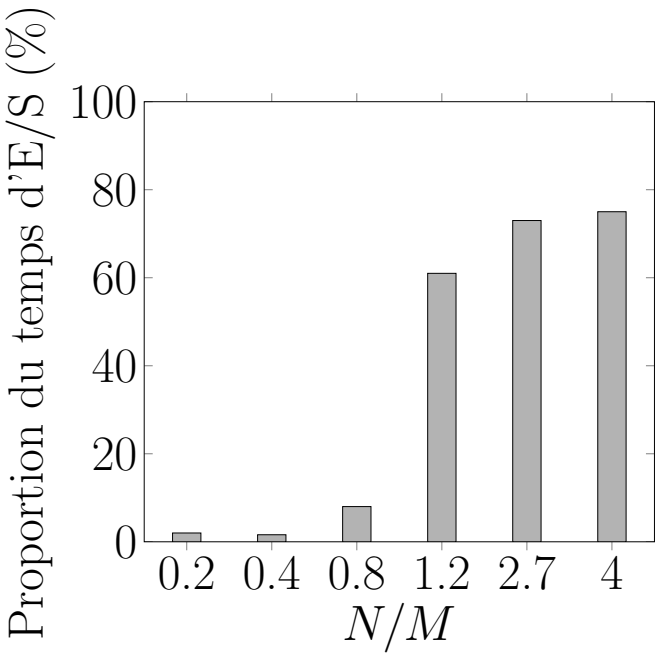


FIGURE 3.1 – Proportion du temps d'E/S par rapport au temps d'exécution total de l'algorithme *K-means*

3.1.2 Analyse expérimentale des E/S de l'algorithme *K-means*

L'objectif est d'évaluer par la mesure la proportion du temps d'E/S par rapport au temps d'exécution total lorsque la taille du *data-set* augmente par rapport à l'espace mémoire disponible.

Méthodologie expérimentale : nous effectuons cette expérimentation sur la plateforme BeagleBone Black décrite en annexe A et nous limitons l'espace mémoire utilisable à 256Mo. Dans cette expérience, nous utilisons des contraintes mémoire de plus en plus élevées. Pour cela, nous utilisons des *data-sets* de différents volumes de manière à avoir les contraintes mémoire $N/M = \{0.2, 0.4, 0.8, 1.2, 2.7, 4\}$. Ces *data-sets* sont générés synthétiquement en utilisant l'une des fonctions de la plateforme R [112] prévue à cet effet et décrite dans l'annexe B.

Résultats et discussion : la figure 3.1 montre la proportion de temps passé à faire des E/S par rapport au temps d'exécution total du *K-means* sous différentes contraintes mémoire (N/M). L'expérience montre que lorsque cette proportion est inférieure à 1, ce qui signifie que la mémoire physique peut contenir entièrement le *data-set*, la proportion de temps d'E/S ne dépasse pas 8%. En revanche, lorsque la contrainte mémoire est supérieure à 1, cette proportion est supérieure à 70%.

3.1.3 Discussion

À travers les analyses théoriques et expérimentales des E/S qui résultent des mouvements des données entre la mémoire physique et l'espace *swap*, nous observons que la proportion de temps passé à faire des E/S augmente rapidement lorsque le volume du *data-set* est supérieur au volume de la mémoire disponible.

Par ailleurs, nous remarquons que puisque l'intégralité des données est parcourue à chaque itération de l'algorithme classique, chaque bloc de données est transféré de et vers l'espace *swap*. En somme, l'algorithme classique du *K-means* exhibe une faible localité spatiale et temporelle. Plusieurs travaux de l'état de l'art [57, 34, 128] proposent un traitement basé sur la technique « diviser pour régner » (division-aggrégation) qui permet de ne pas parcourir tout le *data-set* à chaque itération. Néanmoins, les méthodes proposées ne conservent pas la précision des résultats de l'algorithme de base. Par conséquent, dans ce qui suit, nous proposons une nouvelle méthode basée sur le principe division-agrégation qui présente une précision très proche de la précision de l'algorithme *K-means* de base.

3.2 Solution proposée : K-MLIO

Nous présentons dans ce qui suit la méthode *K-Means with a Low I/O* (K-MLIO), dont l'objectif est d'accélérer l'exécution de l'algorithme du *K-means* en réduisant le volume des E/S dû aux mouvements de données entre la mémoire principale et l'espace *swap*. Dans le chapitre 2, nous avons souligné le fait que l'algorithme classique du *K-means* repose sur l'hypothèse que le *data-set* est entièrement chargé en mémoire. Or, dans le contexte d'explosion du volume de données, l'espace *swap* doit inévitablement, être utilisé pour contenir l'entièreté du *data-set*. Plusieurs méthodes de l'état de l'art utilisent le principe « diviser pour régner » pour répondre à cette problématique. Mais nous relevons deux principaux défauts dans ces méthodes : **(1)** elles altèrent la qualité des résultats en comparaison à la méthode classique, **(2)** elles ne dimensionnent pas le problème selon l'espace mémoire disponible et n'évaluent pas l'impact de ce dernier sur l'efficacité de ces méthodes. Dans K-MLIO, nous reprenons la technique de « diviser pour régner » et proposons une méthode qui se rapproche plus de la précision de l'algorithme classique, et nous prenons en compte l'espace mémoire disponible.

3.2.1 Description générale de la méthode proposée

L'idée principale dans K-MLIO est de diviser le *data-set* en parties (*chunks*) pouvant être contenues en mémoire, c'est à dire que chaque *chunk* contient M observations. Le *K-means*

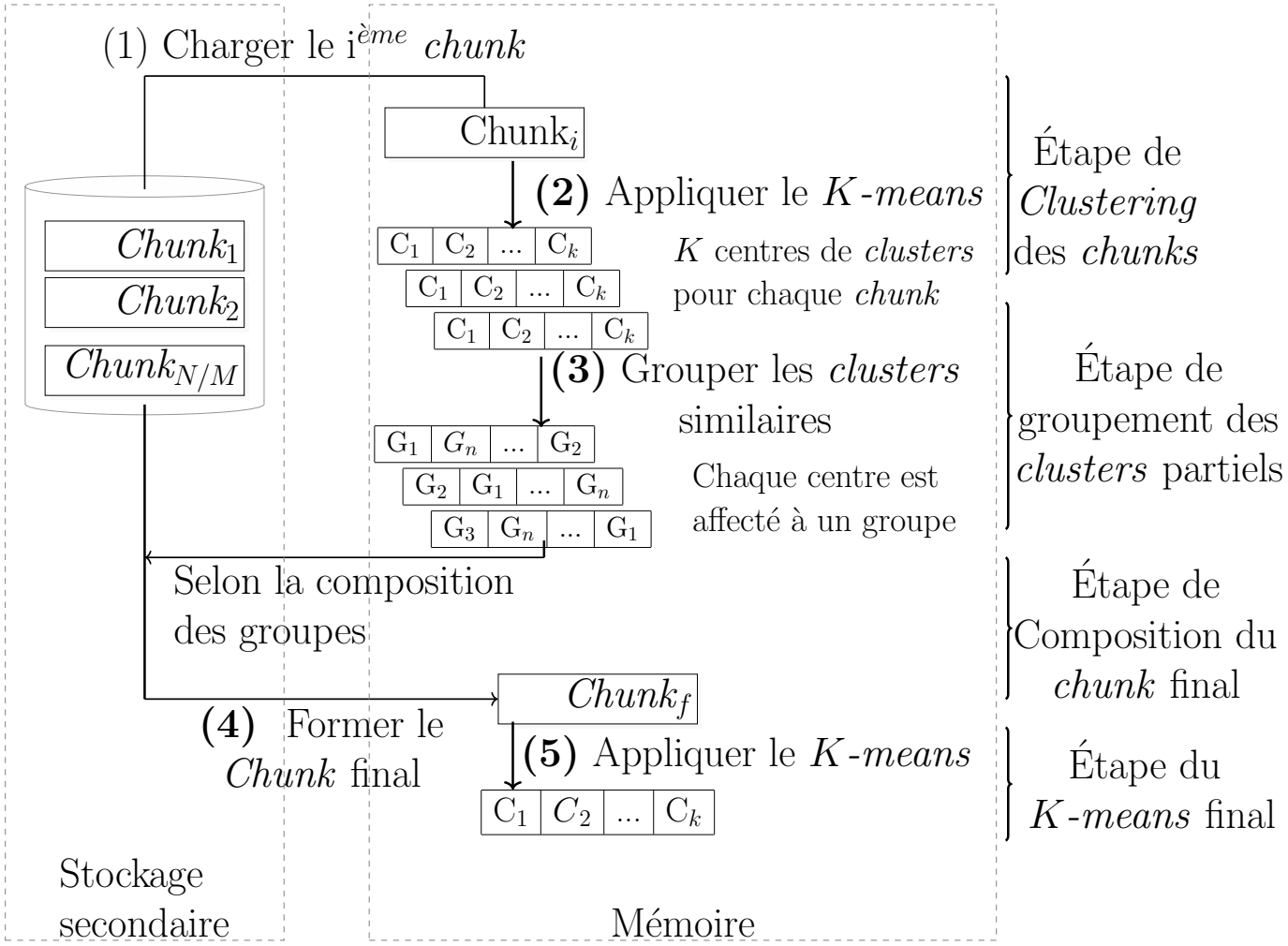


FIGURE 3.2 – Description générale de K-MLIO

est appliqué sur chacun des *chunks* séparément jusqu'à convergence. Grâce aux solutions obtenues pour les *chunks*, nous formons un *chunk* final qui est représentatif du *data-set* en effectuant une analyse de similarité entre tous les *clusters* obtenus. La similarité entre les *clusters* signifie que les observations de l'un peuvent être affectées à l'autre, une définition plus formelle est donnée ci-après (définition 13 page 86). La dernière étape consiste à appliquer le *K-means* sur le *chunk* final pour obtenir le *clustering* final. La figure 3.2 décrit le fonctionnement de K-MLIO selon les étapes suivantes :

1. **Étape de *clustering* des *chunks* :** chaque *chunk* est chargé du stockage secondaire vers la mémoire physique séparément. Le *K-means* lui est appliqué. Les K centres obtenus pour chaque *chunk* sont stockés. Dans la figure 3.2, chaque $chunk_i$ est chargé en mémoire. Le *K-means* lui est appliqué, et les K centres sont gardés en mémoire.
2. **Étape de groupement des *clusters* partiels :** l'objectif de cette étape est de regrouper les *clusters* similaires. Cela permet d'avoir une vue d'ensemble de la distribution des données susceptibles d'appartenir aux mêmes *clusters* sur les *chunks* du *data-set*. La figure 3.2 montre que chaque centre des *clusters* partiels est affecté à un groupe.
3. **Étape de composition du *chunk* final :** il est formé en échantillonnant, de chaque groupe (précédemment formés), un nombre d'observations proportionnel au nombre d'observations du groupe. Dans la figure 3.2, le $chunk_f$ est formé par échantillonnage sur le *data-set*, en se basant sur les groupes obtenus à l'étape précédente.
4. **Étape du *K-means* final :** une fois le *chunk* final formé, le *K-means* lui est appliqué et les centres obtenus sont considérés comme solution au *clustering* de tout le *data-set* (voir la figure 3.2).

3.2.2 Description de la méthode K-MLIO

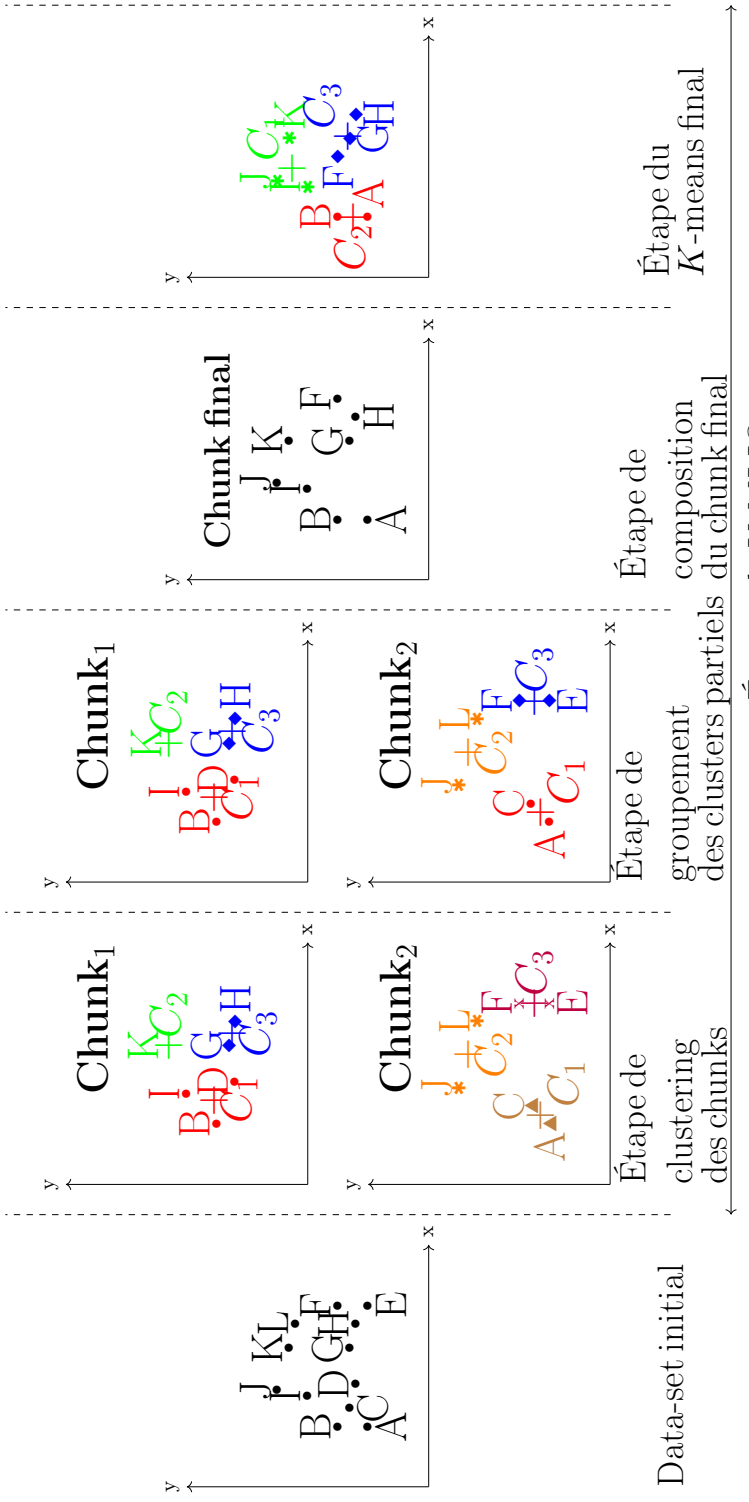


FIGURE 3.3 – Exemple du fonctionnement de K-MLIO

Dans cette section, nous décrivons la méthode K-MLIO et l'illustrons avec un exemple montré dans la figure 3.3. Le *data-set* utilisé dans cet exemple est composé de 12 observations (A à L). Ces dernières sont des points à 2 dimensions. Le nombre de *clusters* à trouver ici est de 3. Le *data-set* utilisé dans cet exemple est le même que celui utilisé pour illustrer le fonctionnement du *K-means* (voir la figure 2.2) pour faciliter la comparaison des fonctionnements des deux algorithmes. Les étapes de K-MLIO sont les suivantes :

— **Étape de *clustering* des *chunks* :** pour un *data-set* D de N observations, si l'on suppose que l'espace mémoire physique disponible peut contenir M observations, alors le nombre de *chunks* à traiter est de $\lceil N/M \rceil$. Pour chacun de ces *chunks*, les observations qu'il contient sont chargées en mémoire, l'algorithme classique du *K-means* est appliqué jusqu'à convergence et les *clusters* formés sont retournés à l'issue de cette étape. En terme d'E/S, l'objectif ici est d'éviter les mouvements de données de chaque bloc du *data-set* entre la mémoire physique et l'espace de *swap* à chaque itération. En effet, le fait d'appliquer le *K-means* sur un nombre d'observations, qui peuvent être contenues en mémoire, permet d'exploiter les données chargées en mémoire sans engendrer d'E/S supplémentaires. Dans l'exemple donné (figure 3.3), le *data-set* d'origine est divisé en deux *chunks* de six observations chacun. Il est à noter que la répartition des observations sur les *chunks* se fait selon leur ordre d'apparition dans le fichier *data-set*. L'application du *K-means* sur les deux *chunks* donne les *clusters* dont l'ensemble des centres est $\{C_1^1, C_2^1, C_3^1\}$ pour le premier *chunk* et $\{C_1^2, C_2^2, C_3^2\}$ pour le deuxième.

— **Étape de groupement des *clusters* partiels :** dans cette étape, une étude de similarité est effectuée entre les *clusters* des différents *chunks*. Nous caractérisons la similarité entre deux *clusters* selon la définition 13 page 86. Le nombre de *clusters* résultants de l'étape précédentes est $K \cdot \lceil N/M \rceil$. En principe, si les observations sont uniformément distribuées sur les *chunks*, alors le nombre de groupes résultants est égale à K. En revanche, dans le pire des cas, où les distributions des données sur les *chunks* sont distinctes les unes des autres, le nombre de groupes est égale à $K \cdot \lceil N/M \rceil$ (la discussion sur la distribution des données sur les *chunks* sera étendue dans la section 3.2.4). Dans l'exemple donné dans la figure 3.3, les *clusters* dont les centres sont C_1^1 et C_1^2 (des *chunks* respectifs 1 et 2) sont similaires puisque la distance entre les deux centres est inférieure aux deux distances moyennes entre les observations et leurs centres respectifs. Par conséquent, les deux *clusters* sont groupés (ils sont montrés avec la même couleur dans la figure 3.3). À la fin de cette étape, on obtient les groupes suivants : $G_1 = \{A, C, B, I, D\}$, $G_2 = \{J, L\}$, $G_3 = \{K\}$ et $G_4 = \{F, E, G, H\}$. En somme, cette étape permet d'avoir une vue globale de la

distribution de données sur le *data-set*. Par exemple, les observations B et C qui sont dans des *chunks* différents sont susceptibles d'appartenir au même *cluster* final puisqu'ils appartiennent au même groupe.

— **Étape de composition du *chunk* final :** l'objectif de cette étape est de former un *chunk* représentatif du *data-set*. Pour ce faire, nous échantillonnons aléatoirement dans chaque groupe (précédemment formé) un ensemble d'observations dont le nombre d'éléments est proportionnel au nombre d'observations du groupe. Si le nombre d'observations du groupe G_i est N_i alors le nombre d'observations à échantillonner de ce groupe est :

$$Sel_i = \lceil N_i / N \cdot M \rceil \tag{3.3}$$

Dans l'exemple donné dans la figure 3.3, les observations sélectionnées du groupe G_1 : sont A, B et I. Une seule observation est sélectionnée du groupe G_2 : J. K est la seule observation sélectionnée du groupe G_3. Trois observations sont sélectionnées du groupe G_4 : G, H et F. À noter que dans cet exemple, 8 observations au total sont sélectionnées pour former le *chunk* final parmi 12, ce qui n'est pas réaliste, mais facilite la compréhension. Autrement dit, le nombre d'observations du *chunk* final est le même que le nombre d'observations que peut contenir la mémoire M.

— **Étape du *K-means* final :** l'algorithme du *K-means* est appliqué sur le *chunk* précédemment formé qui est représentatif du *data-set* de départ. Les résultats retournés par cette étape sont considérés comme les centres des *clusters* finaux.

Définition 13. *Soient C_1 et C_2 deux clusters dont les centres sont respectivement $\vec{\mu}_1$ et $\vec{\mu}_2$, et dont les distances moyennes entre les centres et les observations qui leurs sont affectées $\{\vec{x}_i\}$ sont données par :*

$$v_1 = \frac{1}{|C_1|} \cdot \sum_{i=1}^{|C_1|} d(\vec{x}_i, \vec{\mu}_1) \tag{3.4}$$

pour le cluster C_1, et par :

$$v_2 = \frac{1}{|C_2|} \cdot \sum_{i=1}^{|C_2|} d(\vec{x}_i, \vec{\mu}_2) \tag{3.5}$$

pour le cluster C_2.

On dit que les clusters C_1 et C_2 sont similaires si :

$$d(\vec{\mu}_1, \vec{\mu}_2) \leq v_1 \ \ \& \ \ d(\vec{\mu}_1, \vec{\mu}_2) \leq v_2 \tag{3.6}$$

Ces deux conditions signifient que l'élément représentatif des observations de C_1 ($\vec{\mu}_1$) peut faire

partie du cluster C_2 et l'élément représentatif de C_2 ($\vec{\mu_2}$) peut faire partie du cluster C_1.

3.2.3 Analyse des E/S de K-MLIO

Dans ce qui suit, nous analysons le coût en E/S de K-MLIO et le comparons au coût de la méthode classique analysée en section 3.1.1. Parmi les quatre étapes de K-MLIO, les étapes (1) et (3) sont les plus coûteuses en E/S. En effet, l'étape de groupement des *clusters* est basée sur la comparaison des distances entre les centres, et moyennes respectives des distances intra-*clusters* qui sont calculées pendant l'étape (1). L'étape (4), quant à elle, consiste simplement à appliquer le *K-means* sur le *chunk* final qui est formé et chargé en mémoire à l'étape (3). Les étapes (1) et (3) engendrent les volumes d'E/S suivants :

— **Étape de *clustering* des *chunks* :** chaque *chunk* est chargé depuis l'espace de stockage secondaire une et une seule fois, le coût de ce chargement est donné par :

$$I/O_{cost}^{(1)} = \lceil N/M \rceil \cdot \lfloor M/B \rfloor \tag{3.7}$$

— **Étape de composition du *chunk* final :** dans cette étape, les M observations qui constituent le *chunk* final sont chargées en mémoire. Un index de l'emplacement des *chunks* sur le dispositif de stockage secondaire est formé. Par ailleurs, les observations sélectionnées qui se trouvent dans les mêmes *chunks* sont chargées au même temps pour ne pas parcourir les mêmes *chunks* plusieurs fois. Le meilleur des cas en terme d'E/S survient lorsque toutes les observations à sélectionner se trouvent dans le même *chunk*. Le coût de chargement dans ce cas est :

$$I/O_{cost}^{(2)} = \lfloor M/B \rfloor \tag{3.8}$$

Le pire cas se produit lorsque chacun des *chunks* contient des observations sélectionnées, et donc doit être parcouru. Cela engendre un parcours complet du *data-set*.

$$I/O_{cost}^{(2)} = \lceil N/M \rceil \cdot \lfloor M/B \rfloor \tag{3.9}$$

Le coût total en E/S de K-MLIO est :

$$I/O_{cost}^{(C)} = I/O_{cost}^{(1)} + I/O_{cost}^{(2)} \tag{3.10}$$

Par conséquent, et selon les équations (3.8) et (3.9), le coût en E/S de K-MLIO peut être borné de la façon suivante :

$$\left(\left\lceil \frac{N}{M} \right\rceil + 1\right) \cdot \left\lfloor \frac{M}{B} \right\rfloor \leq I/O_{cost}^{(C)} \leq 2 \cdot \left\lceil \frac{N}{M} \right\rceil \cdot \left\lfloor \frac{M}{B} \right\rfloor \tag{3.11}$$

L'équation (3.11) montre que contrairement à l'algorithme classique du *K-means*, le coût en E/S de K-MLIO ne dépend pas du nombre d'itérations I. La réduction du coût des E/S réalisée par K-MLIO par rapport à la méthode classique du *K-means*, dont le coût est symbolisé par $I/O_{cost}^{(C)}$, est donnée par :

$$R_{I/O} = \frac{I/O_{cost}^{(C)} - I/O_{cost}}{I/O_{cost}^{(C)}} \tag{3.12}$$

En utilisant l'encadrement donné dans l'équation (3.11), on obtient un encadrement pour la réduction du coût en E/S réalisé par K-MLIO par rapport à la méthode classique.

$$1 - \frac{2}{I} \leq R_{I/O} \leq 1 - \frac{1}{I} \cdot \left(1 + \frac{1}{\lceil N/M \rceil}\right) \tag{3.13}$$

L'équation (3.13) montre que la réduction réalisée par K-MLIO en terme d'E/S par rapport au *K-means* classique dépend du volume du *data-set* par rapport à l'espace mémoire disponible $\lceil N/M \rceil$, et du nombre d'itérations nécessaires au *K-means* classique. Plus grands sont ces deux paramètres, meilleure est l'efficacité de K-MLIO comparée à la méthode classique.

3.2.4 Cas particuliers de K-MLIO

Nous avons identifié deux cas de figures où l'application de K-MLIO pourrait dégrader les performances par rapport à l'algorithme classique du *K-means*. Dans ce qui suit, nous mettons en évidence ces cas, et nous proposons des solutions pour adapter K-MLIO.

K-MLIO sur de petits *data-sets*

K-MLIO est conçu pour réduire les E/S causées par les mouvements des blocs du *data-set* entre la mémoire physique et l'espace de *swap*. Ces mouvements de données se produisent lorsque le volume du *data-set* est supérieur au volume de mémoire physique disponible. Lorsque la mémoire physique peut entièrement contenir le *data-set*, l'appli-

cation K-MLIO dégrade les performances, car l'algorithme du *K-means* est appliqué plusieurs fois, ce qui augmente la charge de travail du CPU (dû aux calculs de distances des observations par rapport aux centres). Par conséquent, K-MLIO dans ce cas, est plus gourmand en calcul que l'algorithme classique du *K-means* alors que ce dernier ne cause pas de mouvements de données entre la mémoire principale et l'espace de *swap* puisque le *data-set* est entièrement en mémoire.

Solution : afin de répondre à ce scénario, nous adaptons K-MLIO de façon à ce que l'optimisation en terme d'E/S ne soit appliquée que si le volume du *data-set* est supérieur à l'espace mémoire disponible. Si ce n'est pas le cas, la version classique du *K-means* est appliquée.

Cas de distribution de données non-uniforme

La distribution uniforme des données signifie que les *chunks* du *data-set* contiennent tous, plus ou moins, la même proportion d'observations d'un même *cluster* (c'est à dire que les observations d'un même *cluster* sont distribuées de manière uniforme sur les *chunks*). Le pire cas de distribution non-uniforme se présente lorsque toutes les observations d'un même *cluster* se retrouvent dans un même *chunk*. Dans ce cas, l'étape de *clustering* des *chunks* sera coûteux en terme de calculs. En effet, l'application du *K-means* sur un *chunk* constitué entièrement d'observations qui appartiennent au même *cluster* force à répartir ces observations, qui sont naturellement proches les unes des autres, dans *K clusters*. Par conséquent le nombre d'itérations nécessaires pour converger augmente, ce qui rend l'opération coûteuse en calculs. En revanche, une telle distribution des observations n'affecte pas la qualité des résultats retournés puisqu'à l'étape de groupement des *clusters* similaires, les *clusters* formés à chaque étape se retrouvent dans les mêmes groupes.

Solution : ***K-Means with a Low I/O with Randomization*** **(K-MLIOR).** Pour répondre à ce problème, nous proposons une phase de pré-traitement du *data-set* qui vise à uniformiser la distribution des données avant d'appliquer la méthode K-MLIO. Pour cela, nous procédons par randomisation de manière à redistribuer les observations d'un même *cluster* sur les *chunks*. Le processus de randomisation peut être décrit comme suit : nous définissons une taille de bloc de randomisation R_b et nous créons un index des positions des différents blocs sur l'espace de stockage. Par conséquent, chaque *chunk* est composé de $\lfloor M/R_b \rfloor$ blocs tirés aléatoirement du *data-set*. Ainsi, pour chaque *chunk*, nous maintenons une liste des blocs de randomisation qui le composent. Au moment du chargement du *chunk*, les blocs qui lui sont associés sont lus. Comme l'opération de lecture aléatoire

n'est pas plus coûteuse que la lecture séquentielle sur la mémoire flash [15] (toujours sous l'hypothèse que notre espace de stockage est basé sur la mémoire flash), la phase de randomisation n'est pas coûteuse en temps.

3.3 Évaluation expérimentale

Dans cette section, nous procédons à l'évaluation de la méthode K-MLIO. Nous commençons par décrire la méthodologie expérimentale, puis nous donnons les résultats d'évaluation et leur analyse.

3.3.1 Méthodologie expérimentale

Description des expériences

Expérience 1 : son objectif est d'évaluer l'efficacité de K-MLIO par rapport à l'algorithme d'origine du *K-means*. Pour cela, nous utilisons différentes contraintes mémoire en effectuant les mesures avec des *data-sets* de plus en plus grands (voir la section *data-sets* utilisés). Pour cette expérience, nous faisons également varier l'index de séparation entre les *clusters* du *data-set* qui agit sur la complexité du problème, car plus l'index est élevé, plus les *clusters* sont séparés et donc moindre est le nombre d'itérations nécessaires à la convergence de l'algorithme de *clustering*. La variation de l'index de séparation a pour objectif d'observer l'effet de complexité du problème sur l'efficacité de K-MLIO.

Expérience 2 : dans cette expérience, nous comparons K-MLIO à la méthode *Mini-Batch K-means* [128] (voir la section 8 page 70 pour les détails de cette méthode) qui est largement adoptée pour sa rapidité. Dans la méthode *Mini-Batch K-means*, le nombre d'itérations est fixé a priori, alors que le *clustering* dans K-MLIO itère jusqu'à convergence. Par conséquent, pour obtenir une comparaison pertinente, nous avons commencé par mesurer le temps d'exécution de K-MLIO. Selon le temps obtenu, nous fixons, de manière expérimentale, le nombre d'itérations de *Mini-Btach K-means* de manière à obtenir le même temps d'exécution. La métrique utilisée pour la comparaison est alors la précision obtenue pour des exécutions des deux algorithmes à temps égal.

Expérience 3 : l'objectif de cette méthode est de montrer l'intérêt d'utiliser K-MLIOR lorsque la distribution des données n'est pas uniforme. Pour cela, nous avons comparé les performances de *K-means*, K-MLIO et K-MLIOR sous différentes contraintes et avec un *data-set* dont les observations suivent le pire cas de distribution : toutes les observations

d'un même *cluster* sont dans le même *chunk*. La taille du bloc de randomisation a été fixée à $R_b = 512$ observations.

Expérience 4 : cette dernière expérience a pour objectif de comparer les réductions de proportions de temps d'E/S obtenus par K-MLIO et K-MLIOR par rapport au *K-means*. Cette expérience est réalisée en utilisant le *data-set* synthétique de séparation moyenne 0.01.

Métriques mesurées

Nous décrivons dans cette section les métriques que nous utilisons pour évaluer la méthode que nous proposons.

Métriques temporelles : nous avons mesuré le temps d'exécution de chacun des algorithmes K-MLIO, K-MLIOR, *K-means* et *Mini-Batch K-means*. Par ailleurs, nous avons également mesuré le temps consacré aux E/S afin d'en déduire la proportion de temps passé à faire des E/S par rapport au temps d'exécution total. Chaque expérience est répétée 10 fois pour présenter une moyenne des mesures obtenues.

La réduction du temps d'exécution est calculée par rapport au temps d'exécution de l'algorithme *K-means* de la manière suivante :

$$R = \frac{T_{K-means} - T_{K-MLIO}}{T_{K-means}} \tag{3.14}$$

Métrique de précision : afin de comparer la précision des algorithmes, nous utilisons la métrique de l'erreur moyenne [101] qui est la moyenne des distances entre chaque centre obtenu avec les algorithmes K-MLIO, K-MLIOR, *K-means* et *Mini-Batch K-means* et le centre réel qui lui correspond obtenu à la génération du *data-set*. En effet, pour chaque *data-set* généré, la liste des centres exactes qui correspondent à ses *clusters* est retournée (plus d'explications sur la génération de *data-sets* synthétiques sont données dans l'annexe B). Nous formons K paires, chacune formée par le centre réel $\vec{\mathrm{M}}_i$ et le centre obtenu en appliquant l'algorithme de *clustering* $\vec{\mu}_i$, qui lui est le plus proche. L'erreur est, alors, calculée comme suit :

$$e_i = \frac{1}{d} \cdot \sum_{l=1}^{d} \frac{|\mu_i^l - \mathrm{M}_i^l|}{|\mu_i^l|} \tag{3.15}$$

Taille du *Data-set* (Mo)	100	400	700	1024
Contrainte mémoire N/M	0.4	1.5	2.7	4

TABLE 3.1 – Contraintes mémoire utilisées pour les expériences

Où d est la dimension du *data-set*.

L'erreur moyenne pour l'ensemble des centres obtenus est calculée comme suit :

$$E = \frac{1}{K} \cdot \sum_{i=1}^{K} e_i \tag{3.16}$$

Nous considérons que l'erreur est acceptable si $E \leq 1$ [101].

Data-sets utilisés

Les expériences décrites précédemment ont été exécutées sur des *data-sets* synthétiques générés avec le paquet *ClusterGeneration* de la plateforme R [111]. Nous fixons le nombre K, de *clusters* à générer dans chaque *data-set*, à 10 et le nombre de dimensions d à 10. Nous faisons varier le nombre d'observations M à inclure dans chaque *cluster* de manière à obtenir les contraintes mémoire N/M données dans le tableau 3.1. L'index de séparation des *clusters* d'un *data-set sepval* est fixé à -0.6 pour des *clusters* peu séparés, 0.01 pour des *clusters* moyennement séparés et 0.6 pour des *clusters* à forte séparation.

Plateforme d'exécution utilisée

La plateforme utilisée est la carte de développement embarquée BeagleBone Black. Nous avons créé un espace de *swap* de 2 Go sur une carte SD intégrée à la plateforme. La description technique de cette carte est donnée en annexe A.

3.3.2 Résultats et discussion

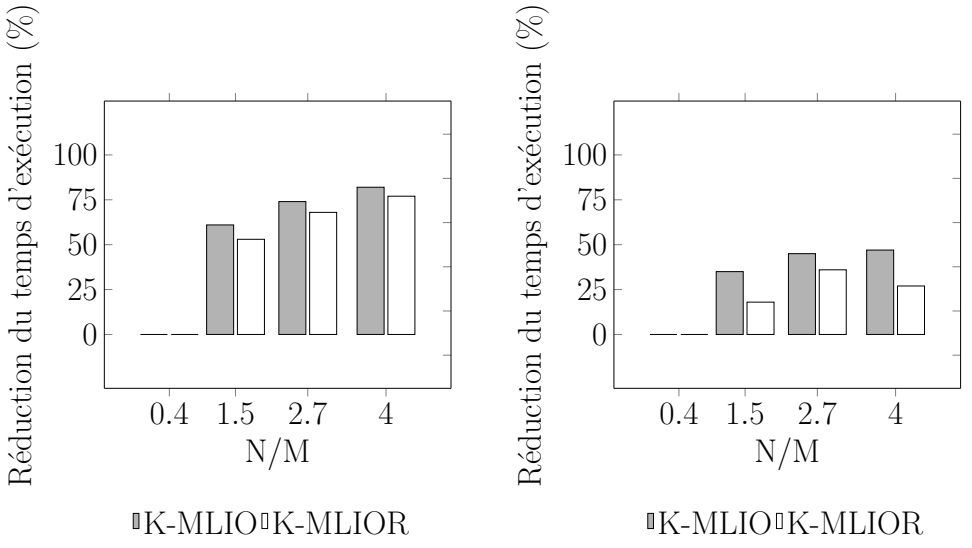
Expérience 1

Le tableau 3.2 montre les erreurs obtenues avec les trois algorithmes K-MLIO et *K-means* pour tous les *data-sets* testés. D'après les résultats donnés dans ce tableau, les erreurs des algorithmes K-MLIO, K-MLIOR [1] et *K-means* sont comparables.

1. Les erreurs de K-MLIO et K-MLIOR sont confondues car la randomisation n'affecte pas la qualité des résultats retournés.

Sepval	N/M	Erreur	
		K-means	K-MLIO/K-MLIOR
-0.6	0.4	6.54	6.54
	1.5	5.7	5.9
	2.7	3.86	3.93
	4	6.2	5.28
0.01	0.4	0.22	0.22
	1.5	0.24	0.15
	2.7	0.11	0.12
	4	0.12	0.12
0.6	0.4	0.003	0.003
	1.5	0.0014	0.007
	2.7	0.0007	0.0036
	4	0.003	0.01

TABLE 3.2 – Erreurs moyennes obtenues avec l'expérience 1



(a) $sepval = 0.01$ (b) $sepval = 0.6$

FIGURE 3.4 – Expérience 1 : réductions de temps d'exécution obtenues avec K-MLIO et K-MLIOR

N/M	Erreur moyenne	
	K-MLIO	*Mini-Batch K-means*
0.4	0.22	0.6
1.5	0.15	0.91
2.7	0.12	0.58
4	0.12	0.59

TABLE 3.3 – Erreur moyenne pour l'expérience 2

Nous montrons dans les figures 3.4a et 3.4b les réductions de temps d'exécution obtenus avec K-MLIO et K-MLIOR avec les index de séparation $Sepval = 0.01$ et $Sepval = 0.6$ par rapport au temps d'exécution du *K-means*. On observe que K-MLIO et K-MLIOR réduisent le temps d'exécution par rapport au *K-means* respectivement de 60 à 82% et 50 à 80% pour $sepval = 0.01$, et 35 à 50% et 20 à 40% pour $sepval = 0.6$ lorsque le volume des *data-sets* est supérieur au volume de mémoire disponible. Nous ne montrons la réduction du temps d'exécution que lorsque l'erreur est acceptable ($E \leq 1$). Or, d'après le tableau 3.2, nous remarquons que lorsque $Sepval = -0.6$ les deux algorithmes ne convergent pas ($E > 1$). On remarque que les réductions de temps d'exécutions sont plus élevées pour les *data-set* dont la séparation entre les *clusters* est moyenne ($Sepval = 0.01$), qu'avec le *data-set* dont la séparation est forte ($Sepval = 0.6$). En effet, si les *clusters* sont très séparés les uns des autres, l'algorithme de *clustering* converge plus vite (le nombre d'itérations nécessaires à la convergence est plus petit). L'équation (3.13) montre que plus le nombre d'itérations est élevé dans l'algorithme *K-means*, plus la réduction du volume d'E/S réalisée est importante, car le nombre de parcours du *data-set* effectués par l'algorithme *K-means* est plus grand.

La deuxième observation que l'on peut faire de ces résultats est que la réduction réalisée par K-MLIO est légèrement plus élevée que celle réalisée avec K-MLIOR. En effet, la phase de randomisation de K-MLIOR qui consiste à former un index qui associe à un bloc de randomisation un *chunk*, augmente légèrement le temps d'exécution par rapport à K-MLIO. Par conséquent, pour un *data-set* dont la distribution est uniforme, la performance de K-MLIO est meilleure que celle de K-MLIOR.

Expérience 2

Dans cette deuxième expérience, nous comparons la précision de K-MLIO par rapport à l'algorithme *Mini-Batch K-means* pour des temps d'exécution égaux. Les résultats sont données dans le tableau 3.3. On remarque que l'erreur moyenne obtenue avec K-MLIO est toujours plus basse que celle de *Mini-Batch K-means*. La raison de cette différence de

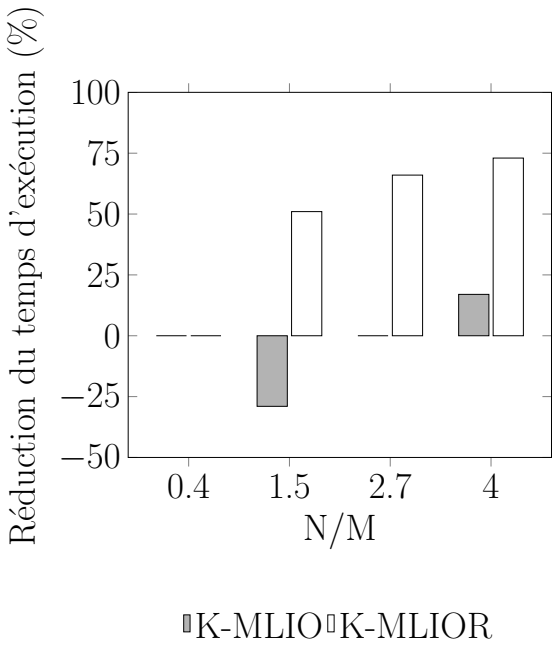


FIGURE 3.5 – Expérience 3 : comparaison des réductions des temps d'exécutions obtenues avec K-MLIO et K-MLIOR

précision pour un même temps d'exécution est que dans le cas de *Mini-Batch K-means*, les sous-ensembles sont formés de manière aléatoire. Par conséquent, la probabilité pour que toutes les observations du *data-set* aient été prises en compte autant de fois que nécessaire est faible, alors que dans le cas de K-MLIO, puisque les *chunks* représentent des séquences de M observations adjacentes, toutes les observations sont prises en compte.

Expérience 3

La figure 3.5 montre les réductions de temps d'exécution obtenues avec les méthodes K-MLIO et K-MLIOR par rapport à l'algorithme classique du *K-means* lorsque le *data-set* suit le pire cas de distribution des données, c'est-à-dire, les observations qui appartiennent aux mêmes *clusters* se retrouvent dans les mêmes *chunks*. D'après la figure, on observe que pour $N/M = 1.5$, K-MLIO est moins performant que *K-means*. En effet, comme les observations appartenant au même *chunk* sont proches les unes des autres, le nombre d'itérations nécessaires pour que les centres ne varient pas d'une itération à l'autre est grand, ce qui augmente le temps d'application du *K-means* sur un *chunk*. À partir de $N/M = 2.7$, K-MLIO obtient de meilleures performances que le *K-means*, car l'augmentation du coût des E/S rallonge le temps d'exécution. K-MLIO devient alors profitable malgré le coût en calculs. On observe que K-MLIOR en revanche, réduit le temps d'application du *K-means* de 51 à 73%, ce qui montre que la méthode de randomisation est

N/M	Erreur moyenne		
	K-means	K-MLIO	K-MLIOR
0.4	0.22	0.22	0.22
1.5	0.2	0.15	0.15
2.5	0.11	0.12	0.12
4	0.12	0.18	0.12

TABLE 3.4 – Erreur moyenne pour l'expérience 3

N/M	Proportion du temps d'E/S (%)		Réduction du temps d'E/S réalisée par K-MLIO (%)
	K-means	K-MLIO	
0.4	1.6	1.6	0
1.5	67	3.5	94
2.7	73	3.5	95
4	75	3	96

TABLE 3.5 – Expérience 4 : réduction du temps d'E/S réalisée par K-MLIO

efficace dans le cas des distributions non-uniformes. En terme de précision, le tableau 3.4 montre que les erreurs moyennes de K-MLIO, K-MLIOR et *K-means* sont comparables, ce qui montre que la randomisation n'altère pas les résultats du *clustering*.

Expérience 4

Dans cette expérience, nous comparons les proportions de temps d'E/S de K-MLIO et du *K-means*. Le tableau 3.5 montre la proportion du temps d'E/S par rapport aux temps totaux d'exécution des deux méthodes, ainsi que la réduction du temps d'E/S réalisée par K-MLIO par rapport au *K-means*. Nous remarquons que la proportion du temps d'E/S ne dépasse pas 3.5% dans le cas de K-MLIO quelle que soit la contrainte mémoire, alors qu'elle augmente rapidement pour l'algorithme *K-means* et atteint 75% lorsque la contrainte mémoire $N/M = 4$. La méthode K-MLIO réduit la proportion du temps d'E/S de plus de 90% pour l'ensemble des tests réalisés. Il est à noter qu'en terme de nombre d'opérations d'E/S, K-MLIO et K-MLIOR sont similaires. La différence entre les deux est le fait que dans le cas de K-MLIO, les accès sont séquentiels, alors que dans K-MLIOR, ils sont aléatoires, ce qui dans le cas d'un espace de stockage basé sur la technologie SSD, n'affecte pas le temps d'accès.

3.4 Conclusion

Dans ce chapitre, nous nous penchons sur le premier cas d'étude d'algorithme de ML à adapter pour l'EI choisi pour cette thèse. Nous proposons une adaptation de l'algorithme *K-means* pour de gros volumes de données par rapport à l'espace mémoire disponible.

Cette adaptation est motivée par le fait que lorsque le volume de données est supérieur à l'espace mémoire disponible, une proportion importante du temps d'exécution est due aux mouvements de données entre la mémoire principale et l'espace de *swap*, car l'algorithme classique du *K-means* effectue plusieurs parcours du *data-set*.

Nous proposons la méthode *K-Means with a Low I/O* (K-MLIO), qui a pour objectif de baisser le volume des E/S dues à ces mouvements. Elle est basée sur le principe « diviser pour régner », qui consiste à diviser le *data-set* en plusieurs *chunks* pouvant être contenus en mémoire, sous l'hypothèse que les observations appartenant à un même *cluster* sont distribuées uniformément sur les *chunks*. Le *K-means* est appliqué sur chacun des *chunks* séparément pour former *K clusters*. L'hypothèse de la distribution uniforme permet de conjecturer qu'en comparant les *clusters* obtenus des différents *chunks*, il existe des groupes de *clusters* similaires, où deux *clusters* C_1 et C_2 sont dits similaires, si les observations de C_1 peuvent être affectées à C_2 et inversement. Par conséquent, nous effectuons ce regroupement des *clusters* similaires afin de connaître la répartition des observations susceptibles d'appartenir aux mêmes *clusters* sur les *chunks*. À partir de ces résultats, nous formons un *chunk* représentatif du *data-set*, en échantillonnant dans chaque groupe un sous-ensemble d'observations. Le *K-means* est alors appliqué à ce *chunk* représentatif pour obtenir le *clustering* final du *data-set*. Les étapes de groupement des *clusters* similaires et de formation du *chunk* représentatif permettent de prendre en considération plus d'observations du *data-set* par rapport aux méthodes de l'état-de-l'art basées sur le principe « diviser mieux régner » et par conséquent, d'augmenter la précision des résultats obtenus.

Notre évaluation montre que cette méthode permet de réduire le temps d'exécution du *K-means* de 80% en préservant la précision des résultats. La comparaison par rapport à la méthode *Mini-Batch K-means* montre qu'à temps égal, K-MLIO est plus précis que *Mini-Batch K-means*.

Afin de répondre au cas où la distribution des observations sur le *data-set* n'est pas uniforme, et en prenant en compte le pire cas de distribution où toutes les observations qui appartiennent au même *cluster* appartiennent au même *chunk*, nous proposons la méthode K-MLIOR qui consiste à randomiser les données avant de procéder au *clustering*. L'évaluation de cette méthode montre qu'elle permet de réduire le temps d'exécution de 51 à 73% par rapport au *K-means* classique dans le cas d'une distribution non-uniforme.

En somme, cette méthode permet de réduire le volume des E/S dus aux opérations de *swap*, et par conséquent de réduire le temps d'exécution du *clustering*. Cette optimisation est intéressante dans le cadre de l'EI puisque de gros volume de données sont amenés à être traités sur des plateformes où l'espace mémoire est restreint, mais également sur des plateformes plus classiques tel que le *cloud* où il est intéressant de réduire l'utilisation des ressources afin de réduire les coûts.

Par ailleurs, la méthode K-MLIO se prête facilement à la parallélisation. En effet, l'application du *K-means* sur les *chunks* peut s'effectuer de manière parallèle. L'étape de groupement des *clusters* similaires s'effectue après obtention des résultats pour chaque *cluster*, pour ensuite former un *chunk* représentatif et effectuer le *clustering* final. Cette parallélisation, peut par exemple, être mise en œuve sur une architecture *cloud*, en utilisant le modèle de programmation *Map-Reduce*, tel que l'opération *Map* applique le *K-means* sur les *chunks*, et *Reduce* aggrège les résultats en formant le *chunk* représentatif sur lequel le *K-means* est appliqué.

Chapitre 4

Algorithme de Random Forests économe en E/S, RaFIO

Introduction

Dans ce chapitre, nous présentons la deuxième contribution de cette thèse, qui porte sur les *Random Forests*. L'objectif de cette contribution est de réduire les E/S qui résultent des mouvements de données entre la mémoire principale et l'espace *swap* lorsque le volume du *data-set* est supérieur au volume mémoire disponible. La synthèse bibliographique a montré que plusieurs travaux ont été proposés pour réduire l'empreinte mémoire de la construction des RF. Comme défini dans la section 2.2.5, la construction d'une RF consiste à entraîner, individuellement, plusieurs arbres de décision, chacun sur la base d'un sous-ensemble d'observations du *data-set*. Un arbre de décision permet de regrouper les observations qui appartiennent aux mêmes classes dans les mêmes nœuds feuilles, et ce, en trouvant la succession de conditions sur les valeurs de propriétés du *data-set* pour obtenir une telle répartition. Cela implique de tester plusieurs propriétés et parcourir plusieurs observations, justifiant ainsi, le caractère intensif des accès mémoire de cet algorithme.

Les mises en œuvre des RF supposent que le *data-set* soit entièrement chargé en mémoire. Or, dans le contexte de l'EI et de l'augmentation du volume de données à traiter, cette hypothèse est difficilement soutenable. Ceci cause les mouvements de données entre la mémoire principale et l'espace de stockage secondaire et augmente le temps de construction d'un arbre de décision. L'une de nos mesures a montré que pour un *data-set* 8 fois plus grand que l'espace mémoire disponible, la construction d'un arbre de décision est 25 fois plus lente que dans le cas d'un espace mémoire supérieur au volume du *data-set*.

Dans ce chapitre, nous identifions les parties de l'algorithme traditionnel responsables des mouvements de données, puis proposons deux révisions de cet algorithme qui permettent de réduire le volume des E/S. Nous clôturons ce chapitre par une évaluation des méthodes proposées et une mise en perspective par rapport aux travaux de l'état de l'art liés aux RF.

Sommaire

4.1 Analyse des accès mémoire des RF

Dans cette section, nous analysons le motif d'accès aux données du *data-set* et les mouvements de données entre la mémoire principale et l'espace de *swap* qui en résultent lors de la construction d'un arbre de décision. Nous commençons par illustrer les mouvements de données en utilisant un exemple simple de construction d'arbre de décision. L'objectif est de mettre en évidence l'origine algorithmique de ces mouvements. Par la suite, nous quantifions ces mouvements de manière théorique avec un modèle simple, puis mesurons leur impact de manière expérimentale.

Label	f_1	f_2	f_3	f_4	Classe
A	$\frac{0}{1}(1)$	$\frac{0}{0}(5)$	$\frac{0}{1}(9)$	$\frac{0}{0}(13)$	$\frac{0}{1}(17)$
B					
C	$\frac{0}{0}(2)$	$\frac{1}{0}(6)$	$\frac{0}{0}(10)$	$\frac{1}{0}(14)$	$\frac{0}{0}(18)$
D					
E	$\frac{1}{0}(3)$	$\frac{1}{1}(7)$	$\frac{1}{1}(11)$	$\frac{1}{0}(15)$	$\frac{1}{1}(19)$
F					
G	$\frac{1}{0}(4)$	$\frac{0}{1}(8)$	$\frac{0}{0}(12)$	$\frac{1}{1}(16)$	$\frac{0}{0}(20)$
H					

TABLE 4.1 – *Data-set* de l'exemple. Chaque ligne du *data-set* représente une observation. Les rectangles représentent des blocs d'E/S et sont numérotés de manière à être utilisés dans les explications.

4.1.1 Motivation par un exemple

Nous supposons un système simplifié [1] composé d'une mémoire principale basée sur la technologie DRAM et une mémoire secondaire, dont un espace *swap* basé sur la technologie Flash. L'algorithme de construction d'un arbre de décision (*Bagging*), suppose que l'espace mémoire peut entièrement contenir le *data-set*. Le *data-set* est chargé en mémoire au début de la construction de la RF et est gardé en mémoire durant toute la construction de la RF.

Soit le *data-set* donné dans le tableau 4.1 sur lequel la RF est entraînée, où $\{f_1, f_2, f_3, f_4\}$ sont les propriétés du *data-set* et *Classe* représente la classe réelle des observations. On suppose que l'espace mémoire disponible peut contenir 4 valeurs ($M = 4$) et qu'un bloc d'E/S peut contenir 2 valeurs ($B = 2$).

La table 4.2 montre les chargements de blocs du *data-set* lors de la division de chaque nœud de l'exemple donné dans la figure 4.1. Pour chaque nœud, nous montrons les observations accédées (la colonne observations), les propriétés à tester pour la division (la colonne propriétés), les blocs chargés pour effectuer la division (le numéro du bloc selon la numérotation donnée dans le tableau 4.1), et la moyenne du pourcentage de données effectivement utilisées par bloc chargé. La première ligne de la table, montre que pour la tentative de division du nœud N_0 selon la propriété f_1, les observations qui doivent être accédées sont : $\{A, A, B, C, C, E, F, H\}$. Par conséquent, les blocs qui doivent être chargés en mémoire sont : $(1), (17), (2), (18), (3), (19), (4), (20)$. Le pourcentage de données utilisées pour le bloc (1) par exemple est de 100%, car il contient des valeurs liées aux ob-

1. Le motif d'accès aux caches du processeur et son optimisation sont hors du champ d'étude de cette thèse.

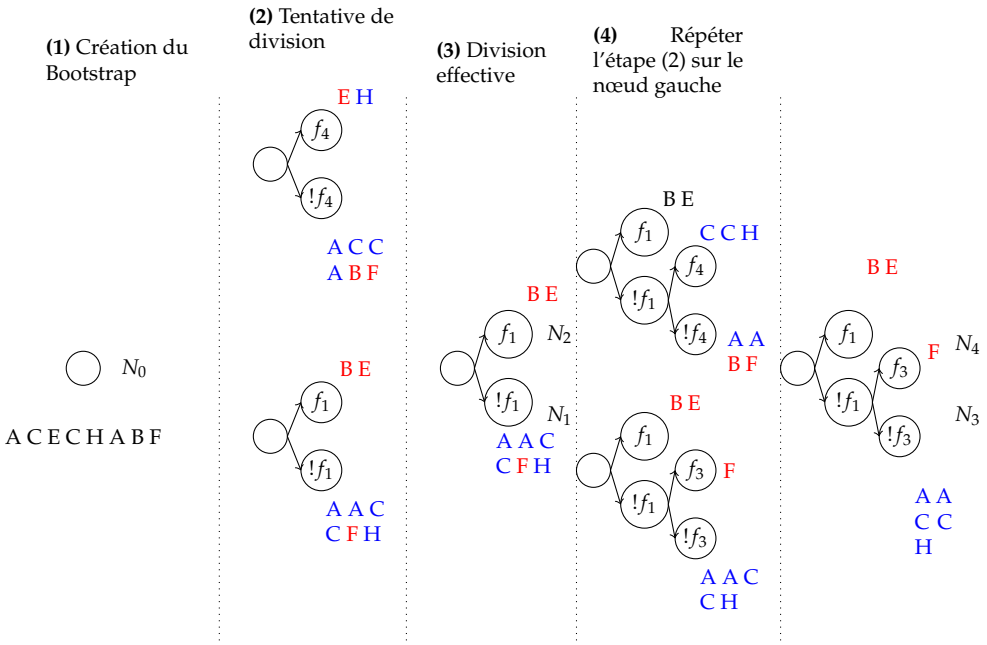


FIGURE 4.1 – Construction d'un arbre de décision

TABLE 4.2 – Mouvements de données lors de la division de chaque nœud du *data-set*

Nœud	Observations	Propriétés	Bloc chargés en mémoire	Pourcentage de données utilisées par bloc
N_0	$\{A, A, B, C, C, E, F, H\}$	f_1	(1), (17), (2), (18), (3), (19), (4), (20)	75%
		f_4	(13), (17), (14), (18), (15), (19), (16), (20)	
N_1	$\{A, A, C, C, F, H\}$	f_3	(1), (17), (2), (18), (3), (19), (4), (20)	50%
		f_4	(13), (17), (14), (18), (15), (19), (16), (20)	
N_2	$\{B, E\}$	/	(2), (18), (3), (19)	50%
N_3	$\{A, A, C, C, F, H\}$	/	(1), (17), (2), (18), (4), (20)	50%
N_4	$\{F\}$	/	(3), (19), (15), (19)	50%

servations A et B, toutes deux faisant partie des observations du nœud N_0. La colonne « Pourcentage de données utilisées par bloc » donne le taux d'utilisation moyen sur tous les blocs accédés lors de la tentative de division (75% dans le cas de N_0 par exemple).

De la table 4.2, nous tirons deux remarques :

1. La faible localité spatiale : lors de la tentative de division du nœud N_1 par exemple, les propriétés à tester sont f_3 et f_4. Les blocs qui contiennent les valeurs de ces propriétés sont chargés, mais exploités à hauteur de 50% (pourcentage de données utiles à la division du nœud). Les 50% de données inutiles sont des observations, soit affectées à d'autres nœuds, soit n'appartenant pas au *bootstrap* (voir la définition d'un *bootstrap* donnée dans la définition 10 page 49). De manière générale, les observations nécessaires à la division d'un nœud sont distribuées sur plusieurs blocs du *data-set*, et ces derniers sont très partiellement exploités à chaque

fois. Par conséquent, l'organisation originale du *data-set* exhibe une faible localité spatiale.

2. Mouvements inutiles de données : le chargement du *data-set* avant le début de la construction des arbres de décision fait que la constitution des blocs se fait selon l'ordre d'apparition des observations dans le fichier du *data-set*. Or, chacun des arbres de décision est construit sur la base d'un *bootstrap* unique. Par conséquent, un bloc peut être chargé de l'espace *swap* vers la mémoire principale alors qu'il contient un seul élément appartenant au *bootstrap*. Dans l'exemple donné dans 4.2, l'observation D n'appartient pas au *bootstrap* mais est chargée en mémoire chaque fois que l'un des blocs (2), (6), (10) ou (14), qui contient des propriétés propres à l'observation C, est chargé.

Cet exemple montre que l'hypothèse d'un *data-set* résident en mémoire et la répartition des observations sur les blocs suivant leur ordre d'apparition dans le fichier du *data-set* est à l'origine des mouvements de données inutiles entre la mémoire principale et le stockage secondaire. Notre intuition est qu'il est possible de réduire le volumes des E/S en s'affranchissant de cette hypothèse.

4.1.2 Analyse théorique des accès mémoire

Dans cette partie, nous analysons, d'un point de vue théorique, le volume d'E/S généré lors de la division d'un nœud.

Au début de la construction d'un arbre de décision, l'intégralité des N observations du *bootstrap* est affectée au nœud racine N_0. Le *bootstrap* étant formé par un échantillonnage avec remise du *data-set* (voir la définition 10 page 49), selon [122, 64] la proportion d'observations du *data-set* susceptibles d'appartenir au *bootstrap* est de 0.63.

L'étape de tentative de division du nœud consiste en la formation des nœuds enfants du nœud selon plusieurs propriétés, puis le choix de la division dont résultent les nœuds enfants les plus pures possible (voir la section 2.2.5).

Pour le nœud racine, le volume théorique d'E/S résultant de la tentative de division selon une propriété est donné par :

$$V_{I/O} = 2 \cdot \lceil 0.63 \cdot N/M \rceil \cdot \lfloor M/B \rfloor \tag{4.1}$$

La proportion $\lceil N/M \rceil$ représente le nombre de parties du *data-set* de tailles égales au nombre d'observations que peut contenir la mémoire disponible. La proportion $\lfloor M/B \rfloor$ représente le nombre de blocs qui constituent la mémoire. Le facteur 2 dans l'équation

résulte du fait suivant : pour effectuer une tentative de division selon une propriété donnée, les blocs qui contiennent cette dernière, mais aussi ceux qui contiennent la classe doivent être accédés.

Or, la distribution des observations sur les blocs dépend de leur ordre d'apparition dans le fichier *data-set*. Par conséquent, les données du *bootstrap* sont susceptibles d'être distribuées de manière uniforme sur les blocs, ce qui se traduit par un accès à tous les blocs du *data-set*. Le volume $V_{I/O}$ est donné alors par :

$$V_{I/O} = 2 \cdot \lceil N/M \rceil \cdot \lfloor M/B \rfloor \tag{4.2}$$

Pour un nœud quelconque auquel sont affectées k observations ($k < N$), le nombre de blocs à charger pour accéder aux k observations dépend de la distribution des données sur le fichier du *data-set*. Le pire cas de distribution se présente lorsque tous les blocs contiennent des observations affectées au nœud. Dans ce cas, le volume d'E/S est donné par :

$$V_{I/O} = 2 \cdot \lceil N/M \rceil \cdot \lfloor M/B \rfloor \tag{4.3}$$

Le meilleur cas de distributions se présente lorsque les k observations du *data-set* sont regroupées dans un sous-ensemble de blocs. Dans ce cas, le volume d'E/S est :

$$V_{I/O} = 2 \cdot \lceil k/M \rceil \cdot \lfloor M/B \rfloor \tag{4.4}$$

Dans les premiers niveaux de l'arbre (proches du nœud racine), le nombre d'observations affectées aux nœuds, k, est proche du nombre d'observations dans le *bootstrap* N. Par conséquent, l'écart entre les volumes d'E/S théorique et réel est faible. En revanche, à mesure que le nœud à diviser se trouve à un niveau profond dans l'arbre, cet écart augmente, ce qui montre qu'un volume important d'E/S est inutile et est dû à la distribution des données sur le fichier du *data-set*.

4.1.3 Analyse expérimentale des accès mémoire

Dans cette section, notre objectif est de mesurer l'impact des E/S dues aux mouvements de données entre l'espace de *swap* et la mémoire principale sur le temps de construction d'un arbre de décision.

Méthodologie expérimentale : l'expérimentation réalisée consiste à mesurer le temps de construction d'un arbre de décision en réduisant de plus en plus le volume mémoire disponible. Nous mesurons également le temps consacré aux E/S lors de cette construction,

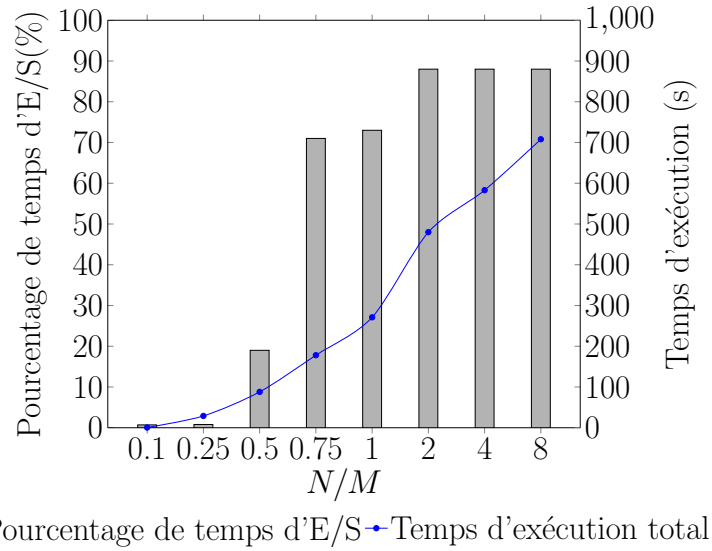


FIGURE 4.2 – proportion du temps d'E/S par rapport au temps total lors de la construction d'un arbre de décision

puis calculons la proportion du temps d'E/S par rapport au temps total de construction. Le *data-set* utilisé pour cette expérience est « *Wearable* » disponible sur le répertoire « *UCI Data-set Repository* » [41]. N/M traduit la proportion du volume du *data-set* par rapport au volume mémoire disponible.

Résultats et discussions : la figure 4.2 montre le temps de construction d'un arbre de décision, ainsi que la proportion du temps d'E/S pour différentes valeurs de N/M. Nous remarquons que plus la contrainte mémoire augmente, plus la proportion du temps d'E/S par rapport au temps d'exécution total augmente. Elle atteint 90% lorsque la taille du *data-set* est 2 fois plus grand que l'espace mémoire. Cette augmentation du temps d'E/S induit une augmentation du temps d'exécution. Dans cette expérience, le temps d'exécution lorsque $N/M = 0.25$ est multiplié par près de 14 lorsque $N/M = 8$.

4.1.4 Synthèse

À travers l'analyse des accès mémoire de l'algorithme de construction d'une RF, nous constatons qu'une partie du volume d'E/S est causée par la faible localité spatiale, autrement dit, les mêmes données sont chargées plusieurs fois. Par ailleurs, le chargement de tout le *data-set* avant le début de construction des arbres de décision est responsable du fait que les blocs de données contiennent des données inutiles, car non incluses dans le *bootstrap*. La méthode de construction peut donc être modifiée de façon à réduire l'effet de ces deux motifs et diminuer le volume des E/S.

4.2 Solution proposée : RaFIO

Dans cette section, nous présentons une méthode de construction de forêts aléatoires économe en E/S, ou *Random Forest I/O-aware algorithm* (RaFIO), dont l'objectif est de réduire le volume des E/S pour les environnements contraints en mémoire. Dans cette section, nous commençons par décrire de manière générale la méthode proposée, en tenant compte des observations faites dans la section 4.1. Nous détaillons, par la suite, les mécanismes employés dans la solution.

4.2.1 Description générale de la méthode

La section 4.1 montre qu'une proportion non négligeable du temps de construction d'un arbre de décision est due aux mouvements de données entre la mémoire principale et l'espace de stockage secondaire, lorsque le volume du *data-set* est supérieur à l'espace mémoire disponible. La méthode proposée est structurée autour de deux optimisations pour répondre aux observations notées dans la section précédente qui sont la faible localité spatiale, et les mouvement des données inutiles. La figure 4.3 donne la description générale de RaFIO, décrite ci-après.

— **Mécanisme de réorganisation du *data-set* :** l'objectif de ce mécanisme est de ré-écrire les données de manière à améliorer la localité spatiale. Pour ce faire, nous proposons de réorganiser les blocs du *data-set* de sorte que chacun d'entre eux contienne des observations susceptibles d'être accédées lors de la division d'un même nœud. Puis, de construire la forêt aléatoire sur le base de ce *data-set* réorganisé. Ce mécanisme est divisé en deux fonctions qui s'exécutent successivement :

  — **Apprentissage de la localité des données :** l'objectif ici est de déduire à partir d'une analyse du *data-set*, les groupes d'observations susceptibles d'être accédées successivement. Le procédé d'apprentissage est décrit dans la section 4.2.2.

  — **Écriture effective des données :** les données apprises par le module d'apprentissage de la localité spatiale sont transmises au module d'écriture du *data-set* tel que montré figure 4.3. Chaque groupe d'observations susceptibles d'être accédées lors de la division d'un même nœud est écrit sur les mêmes blocs de données sur le support de stockage. À la fin du processus, on obtient une copie du *data-set* organisée de sorte à augmenter la localité spatiale. Le *data-set* original peut alors être supprimé ou gardé selon les besoins d'utilisation. Le coût de réécriture du *data-set* est amorti car lu plusieurs fois. Ce coût sera discuté dans

la partie évaluation.

— **Accès aux données à la demande :** l'objectif de cette optimisation est de réduire le volume des E/S dues aux mouvements de données inutiles, en accédant aux données à la demande, au lieu de charger la totalité du *data-set*. Elle est décomposée en 3 modules :

  — **Module de construction d'arbre de décision :** ce module est responsable de la construction effective des arbres de décision selon l'algorithme du *bagging*. À la différence de l'algorithme classique, ce module lève l'hypothèse de présence des données en mémoire principale. Les observations du *data-set* sont chargées à la demande, chaque fois que l'algorithme en a besoin pour diviser un nœud d'arbre de décision. Pour y parvenir, ce module se base sur les deux autres modules décrits ci-après.

  — **Module de *monitoring* de l'espace de travail :** ce module compare le volume du *data-set* (N) par rapport à l'espace mémoire disponible (M). Selon les valeurs de N (transmis par le module de construction d'arbre de décision, voir la figure 4.3) et M, trois scénarios peuvent se dégager : toutes les observations du nœud à diviser peuvent être contenues en mémoire (**scénario 1**) ; la mémoire peut contenir les valeurs de propriétés nécessaires à la division du nœud en cours (**scénario 2**) ; enfin, la mémoire ne peut pas entièrement contenir les propriétés des observations nécessaires à la division d'un seul nœud (**scénario 3**). Ce module retourne le scénario dans lequel l'application se trouve à chaque début de division d'un nœud.

  — **Module de chargement des données :** l'objectif de ce module est de charger les observations utiles à la division d'un nœud en tenant compte du scénario dans lequel l'application se trouve (information transmise par le module moniteur d'espace mémoire, voir la figure 4.3), et en essayant de maximiser la réutilisation des données sur les niveaux inférieurs de l'arbre. Une fois que les données sont chargées, elles sont traitées par le module de construction de l'arbre de décision (voir la figure 4.3). Au démarrage de la construction de la forêt aléatoire, ce module forme un index des positions des données sur le stockage secondaire de façon à réduire les données inutilement lues pour charger les données utiles. Cet index est utilisé tout au long de la construction de la forêt.

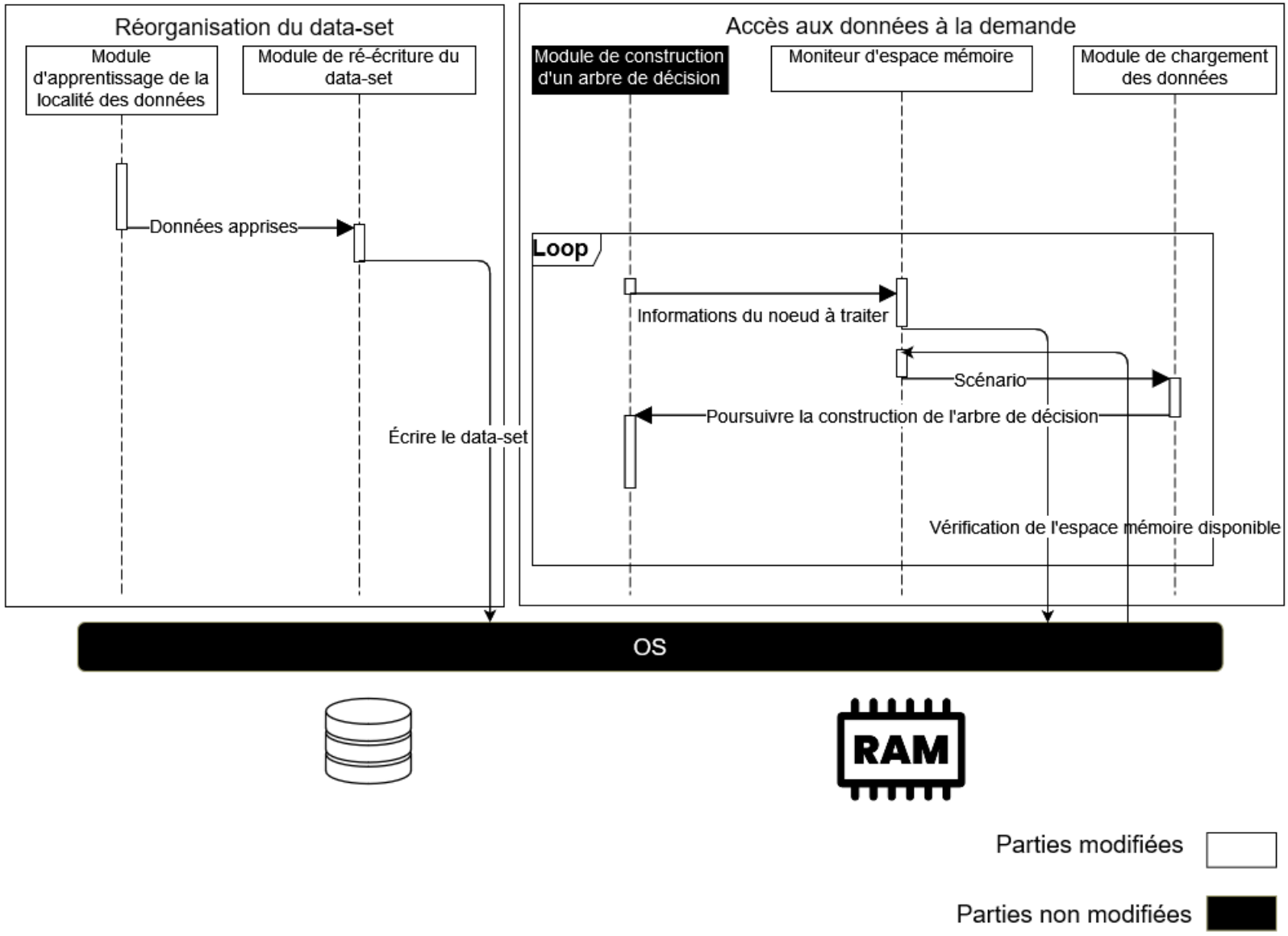


FIGURE 4.3 – Description générale de RaFIO

Il est à noter que les deux optimisations peuvent être employées indépendamment l'une de l'autre pour réduire le volume des E/S.

4.2.2 Réorganisation du *data-set*

La réorganisation des données est basée sur l'observation que les arbres de décision appartenant à une même forêt exhibent des similarités dans leurs manières de classifier les observations du *data-set*. Dans ce qui suit, nous définissons la similarité entre arbres de décision, et la manière d'employer cette propriété pour augmenter la localité des données dans une forêt aléatoire.

Formalisation de la similarité entre arbres de décision

L'objectif dans cette section est de montrer que pour une paire d'observations affectées à un même nœud feuille dans un arbre de décision, la probabilité que ces deux observations soient affectées à un même nœud feuille dans un autre arbre de décision est élevée. Notre intuition, est que, le *data-set* peut être partitionné en sous-ensembles d'observations susceptibles d'être accédées en même temps. Ci-après, nous formalisons cette propriété et évaluons sa pertinence.

Supposons un *bootstrap* qui contient N observations. Soient T_1 et T_2 deux arbres de décision construits sur la base de ce *bootstrap*. Un arbre de décision aboutit à un ensemble de nœuds feuilles. Chacun de ses nœuds feuilles regroupe des observations qui possèdent un certain nombre de propriétés communes. L'union des groupes d'observations affectées aux nœuds feuilles est égale au *bootstrap* de départ, et l'intersection entre ces groupes est un ensemble vide. Cela correspond à un partitionnement[2] (*clustering*) du *bootstrap*, où chaque groupe d'observations associé à un nœud feuille représente un *cluster*. Supposons que l'arbre T_1 (respectivement T_2) induit le partitionnement P_1 (resp. P_2). Pour déduire si une paire d'observations affectées à un même nœud est susceptible d'être accédée en même temps, nous pouvons comparer les partitionnements P_1 et P_2 obtenus par deux arbres de décision.

Il existe dans la littérature plusieurs métriques qui évaluent la similarité entre les partitionnements [61]. Parmi celles-ci, la métrique *Adjusted Rand Index* (ARI) est largement utilisée [125]. Elle dérive de la métrique *Random Index* (RI) qui est calculée comme suit :

$$RI = \frac{a+b}{\binom{N}{2}} \tag{4.5}$$

2. La partition d'un ensemble E est un ensemble de parties non-vides de E, deux à deux disjointes et dont l'union donne E.

X/Y	Y_1	Y_2	...	Y_s	Somme
X_1	n_{11}	n_{12}	...	n_{1s}	a_1
X_2	n_{21}	n_{22}	...	n_{2s}	a_2
...	...	...	...	...	...
X_r	n_{r1}	n_{r2}	...	n_{rs}	a_2
Somme	b_1	b_2	...	b_s	

TABLE 4.3 – Table de contingence

Où a est le nombre de paires d'observations affectées aux mêmes *clusters* dans P_1 et P_2, autrement dit, le nombre de paires d'observations sur lesquelles les partitionnements P_1 et P_2 sont en accord pour regrouper les deux observations dans le même *cluster*. b est le nombre d'observations affectées à des *clusters* différents dans P_1 et P_2, autrement dit, l'accord de P_1 et P_2 pour séparer les deux observations. L'équation traduit, alors, la proportion d'accords des deux partitionnements sur l'affectation d'une paire d'observations par rapport au nombre total de paires d'observations. Le problème que soulève la métrique RI est que lorsque la valeur de b est très élevée par rapport à a, la valeur de RI est proche de 1 sans garantir de similarité entre les partitionnements. En effet, si l'on utilise la métrique RI pour mesurer la similarité de deux partitionnements générés aléatoirement[3], la valeur sera proche de 1, à cause d'une valeur de b élevée.

Pour palier ce problème, la métrique ARI est proposée pour ajuster le RI. Cette métrique est basée sur la matrice de contingence pour deux partitionnements $P_1 = \bigcup_{i=1}^{r} X_i$ et $P_2 = \bigcup_{j=1}^{s} Y_j$ définie dans la table 4.3.

Où $n_{ij} = |X_i \cap Y_j|$, a_i et b_j représentent respectivement le nombre d'observations dans X_i et Y_j. L'indice ARI est calculé selon l'équation suivante :

$$ARI = \frac{\binom{N}{2} \cdot \sum_{i,j}^{r,s} \binom{n_{ij}}{2} - \sum_i^r \binom{a_i}{2} \cdot \sum_j^s \binom{b_j}{2}}{1/2 \cdot \binom{N}{2} \cdot \left[\sum_i^r \binom{a_i}{2} + \sum_j^s \binom{b_j}{2}\right] - \sum_i^r \binom{a_i}{2} \cdot \sum_j^s \binom{b_j}{2}} \tag{4.6}$$

Les valeurs de l'ARI sont entre -1 et 1. L'ARI d'un partitionnement aléatoire est proche de 0 alors qu'un ARI proche de 1 traduit une forte similarité. L'indice ARI peut prendre des valeurs négatives lorsque les partitionnements obtenus sont extrêmement discordants. Cela se produit lorsqu'il y a moins d'accords entre les partitionnements que l'on compare que dans des partitionnements aléatoires [25].

3. La classe de chaque observation est désignée de manière aléatoire.

Mesure expérimentale de l'ARI : afin de vérifier si cette propriété est pertinente dans le cas des partitionnements obtenus avec des arbres de décision, nous avons mesuré les ARI obtenus avec des paires d'arbres de décision. Cette expérimentation a été faite avec plusieurs *data-sets* tirés du dépôt de *data-sets* UCI [41]. Les résultats sont montrés dans la table 4.4. On observe que les valeurs d'ARI obtenues varient entre 0.12 et 0.42, excepté pour le *data-set* Poker. Pour savoir si ces valeurs traduisent une similarité élevée, nous nous référons aux études de l'état de l'art [125, 119]. Dans ces études, plusieurs algorithmes de *clustering* largement utilisés tels que le *K-means*, les GMM, etc. ont été évalués en utilisant la métrique ARI, et les résultats retournés par la mesure de l'ARI par rapport aux partitionnements réels varient entre 0.05 et 0.8 [61]. Par conséquent, nous admettons que les valeurs d'ARI mesurées et données dans la table 4.4 pour les partitionnements générés par deux arbres de décision d'une même forêt aléatoire sont suffisamment élevées pour traduire une similarité des partitionnements.

Le *data-set* Poker est réputé très complexe du fait qu'il est non équilibré et de la manière dont les données y sont représentées [23]. Par conséquent, nous expliquons la faible similarité entre les deux arbres obtenus par la complexité de ce *data-set*.

TABLE 4.4 – Mesures ARI obtenues

Data-set	Wearable	Adult	Covertype	Ecoli	Wine	Heart Failure	Poker
ARI	0.27	0.26	0.12	0.37	0.42	0.33	0.034

Méthode de réorganisation du *data-set*

La méthode proposée consiste à réorganiser les observations du *data-set* de tel sorte à ce que les observations qui appartiennent aux mêmes blocs de données soient accédées durant la division d'un même nœud. Pour ce faire, nous utilisons la propriété de similarité précédemment détaillée. En effet, puisque dans chaque paire d'arbres de décision extraite de la forêt, les deux arbres sont suffisamment similaires, l'idée est de construire un arbre de décision sur le *data-set* à partir duquel la répartition des observations sur les nœuds feuilles est déduite. Cette information est utilisée par la suite pour réécrire les groupes d'observations susceptibles d'être accédées en même temps sur les mêmes blocs. Cela induit la copie complète du *data-set* d'origine. Toutefois, le coût de cette copie sera amorti par le gain en volume d'E/S réalisé lors de la construction des autres arbres de décision, dont le nombre est fixé, par défaut, à 100 dans *Ranger* [149] et *Scikit-Learn* [105]. Cet amortissement sera montré dans l'évaluation. Les étapes de construction du nouveau *data-set* sont données ci-après et synthétisées dans l'algorithme 8.

1. **Construction de l'arbre T_0 :** cette étape est réalisée par le module d'apprentissage de localité montré dans la figure 4.3. L'objectif de cette étape est de construire un arbre de décision qui permet de déduire les groupes d'observations qui ont une forte probabilité d'être accédées dans la même fenêtre temporelle. La construction de l'arbre T_0 se fait selon l'algorithme classique de construction d'arbre de décision (voir l'algorithme 8), à l'exception des instructions responsables de l'écriture des blocs de données qui sera détaillé dans le paragraphe suivant. La construction de cet arbre de décision aboutit à un partitionnement des observations tel que chaque nœud feuille représente un *cluster* d'observations susceptibles d'être accédées successivement. Pour avoir un partitionnement de l'ensemble des observations du *data-set* et non pas d'un *bootstrap* uniquement, nous construisons l'arbre T_0 sur l'ensemble des observations du *data-set*. Ainsi, l'arbre T_0 est plus profond que le reste des arbres de décision qui vont être construits dans la forêt, puisque classifiant plus d'observations. De plus, l'arbre T_0 est plus sujet au problème d'*overfitting* (voir la définition 8 page 48). Par conséquent, dans la méthode proposée, l'arbre T_0 est un arbre temporaire, qui ne sera pas utilisé lors la phase d'inférence et qui sert uniquement à partitionner les observations.
2. **Réécriture du *data-set* sur le support de stockage :** une fois que l'arbre de décision T_0 est obtenu, nous disposons des groupes d'observations susceptibles d'être accédées en même temps. Cette information est utilisée pour réécrire le *data-set* sur le stockage secondaire. Nous supposons qu'il y a assez d'espace sur le stockage secondaire pour contenir la copie du *data-set*. Le nouveau *data-set* est écrit de sorte que les observations qui appartiennent à un même nœud feuille se retrouvent sur les mêmes blocs de données, ce qui permet d'augmenter la localité spatiale. Concrètement, à chaque fois qu'un nœud feuille est atteint, les observations qui lui sont affectées sont écrites dans un/plusieurs blocs de données (voir les lignes 6-10 de l'algorithme 8). Il est à noter que si la taille d'un bloc ne suffit pas à contenir toutes les observations d'un nœud feuille, plusieurs blocs sont utilisés.
3. **Construction effective de la forêt aléatoire :** une fois que les blocs du *data-set* sont écrits selon le partitionnement résultant de l'arbre T_0, les arbres de décision de la forêt sont construits sur la base du nouveau *data-set*.

Il est à noter que la construction de l'arbre de décision T_0 et l'écriture du contenu d'un nœud feuille se confondent dans notre mise en œuvre comme le montre l'algorithme 8. Les deux étapes ont été délibérément séparées afin d'en simplifier l'explication. L'objectif d'une telle démarche est de libérer l'espace mémoire réservé à l'indexation des observations affectées à un nœud feuille. En terme de coût en E/S, le coût de réorganisation du

data-set équivaut au coût de construction d'un arbre de décision auquel s'ajoute le coût d'écriture du *data-set*.

Algorithm 8: Construction de T_0 et réorganisation du *data-set*

```
Data: Matrice de données
while Il existe un nœud non pure n parmi les nœuds feuilles de l'arbre do
    Créer un sous-ensemble de propriétés F
    for f ← 0 to |F| − 1 do
        Tentative de division du nœud n selon f
    n_l, n_r ← Division effective de n selon la meilleure propriété
    foreach nœud cn dans {n_r, n_l} do
        if cn est pure then
            Écrire les observations du nœud dans les mêmes blocs de données
        else
            Ajouter cn dans la liste des nœuds non pures
```

Illustration du fonctionnement de la réorganisation du *data-set*

Nous illustrons, le fonctionnement de cette méthode en reprenant l'exemple utilisé section 4.1.1.

La construction de l'arbre T_0 se base sur tout le *data-set* et donne l'arbre montré figure 4.4. Les observations d'un nœud feuille sont écrites dans des blocs adjacents. Le nouveau *data-set*, organisé de manière à augmenter la localité spatiale, est donné figure 4.5. On remarque que les observations appartenant aux mêmes nœuds feuilles se retrouvent dans les mêmes blocs ou dans des blocs adjacents.

Le motif d'accès aux blocs de données est montré tableau 4.5 page 115. La division du nœud N_0 nécessite l'accès aux observations $\{A, A, C, C, H, F, E, B\}$. Pour la tentative de division selon la propriété f_1 par exemple, les blocs qui doivent être accédés sont ceux qui contiennent les valeurs de la propriété f_1 des observations du nœud, à savoir les blocs $(1), (2), (3), (4)$ ainsi que les blocs contenant les valeurs des classes des observations : $(17), (18), (19), (20)$. En comparaison au motif d'accès aux blocs de données lorsque le *data-set* d'origine est utilisé (tableau 4.2 page 102), on remarque que le pourcentage moyen de données utilisées par bloc est plus élevé avec la méthode proposée. Pour ce petit exemple qui sert à illustrer le fonctionnement de la méthode, la différence est relativement faible, mais sera notable pour des tailles de blocs plus grandes.

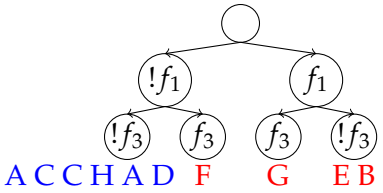


FIGURE 4.4 – Arbre T_0

Label	f_1	f_2	f_3	f_4	Classe
A C	$\frac{0}{0}(1)$	$\frac{0}{1}(5)$	$\frac{0}{0}(9)$	$\frac{0}{1}(13)$	$\frac{0}{0}(17)$
H D	$\frac{0}{0}(2)$	$\frac{1}{0}(6)$	$\frac{0}{0}(10)$	$\frac{1}{0}(14)$	$\frac{0}{0}(18)$
F G	$\frac{0}{1}(3)$	$\frac{1}{0}(7)$	$\frac{1}{0}(11)$	$\frac{0}{1}(15)$	$\frac{1}{0}(19)$
E B	$\frac{1}{1}(4)$	$\frac{1}{0}(8)$	$\frac{1}{1}(12)$	$\frac{1}{0}(16)$	$\frac{1}{1}(20)$

FIGURE 4.5 – *Data-set* réorganisé. Les rectangles représentent des blocs d'E/S et sont numérotés de manière à être utilisés dans les explications

4.2.3 Accès aux données à la demande

Dans cette section, nous détaillons la deuxième optimisation. Nous décrivons succinctement chacun des trois scénarios introduits dans la description générale.

L'objectif des accès à la demande est de réduire les mouvements de données inutiles et par conséquent, le volume d'E/S lors de la construction des arbres de décision. Pour ce faire, seules les données qui sont effectivement demandées lors de la division d'un nœud sont chargées et gardées en mémoire.

Selon le volume mémoire disponible M et le volume des observations assignées au nœud à diviser, le module de *monitoring* mémoire définit, selon l'algorithme 9, le scénario dans lequel l'application se trouve et la méthode de chargement des données.

— Si l'espace mémoire peut contenir toutes les propriétés des observations du nœud, ces données sont chargées, ce qui permet de construire le sous-arbre allant du nœud courant, jusqu'au nœud feuille (scénario 1) ;

— S'il n'y a pas assez de mémoire pour le scénario 1, le module de *monitoring* mémoire vérifie si l'espace mémoire disponible peut contenir le sous-ensemble F des propriétés à tester pour la tentative de division des observations du nœud (scénario 2) ;

TABLE 4.5 – Motif d'accès au *data-set* lors de la division de chaque nœud du *data-set* après réorganisation des données

Nœud	Observations	Propriétés	Bloc accédé	Pourcentage de données utilisées par bloc
N_0	$\{A, A, C, C, H, F, E, B\}$	f_1	(1), (17), (2), (18), (3), (19), (4), (20)	75%
		f_4	(13), (17), (14), (18), (15), (19), (16), (20)	
N_1	$\{A, A, C, C, H, F\}$	f_3	(1), (17), (2), (18), (3), (19)	66.66%
		f_4	(13), (17), (14), (18), (15), (19)	
N_2	$\{B, E\}$	/	(4), (20)	100%
N_3	$\{A, A, C, C, H\}$	/	(1), (17), (2), (18)	75%
N_4	$\{F\}$	/	(3), (19), (15), (19)	50%

— Sinon, on se trouve dans le scénario 3. Dans ce cas, nous proposons de subdiviser les données nécessaires à la tentative de division du nœud en petites parties pouvant être contenues en mémoire, et sur lesquelles la tentative de division se fait indépendamment. Les résultats obtenus pour ces petites parties sont combinés pour aboutir au résultat final. De cette manière, chaque partie de données est chargée une seule fois lors de la tentative de division d'un nœud.

Dans ce qui suit, nous détaillons chacun des scénarios.

Algorithm 9: Chargement des données

Data: N_i : Nœud à traiter, M : Espace mémoire disponible , D : Fichier Data-set, F : sous-ensemble de données à tester

1 **if** $|N_i| \cdot (d+1) \leq M$ **then**
2 Scénario ← scénario 1
3 Lire de D toutes les propriétés des observation de N_i dans la matrice X
4 **else if** $|N_i| \cdot (|F|+1) \leq M$ **then**
5 Scénario ← scénario 2
6 Lire de D toutes les propriétés appartenant à F des observation de N_i dans la matrice X
7 **else**
8 Scénario ← scénario 3

Scénario 1, construction d'un sous-arbre complet

Dans ce scénario, l'espace mémoire disponible peut contenir toutes les propriétés ainsi que la classe effective du nœud courant. Cela se traduit par la condition suivante : $M \geq |N_i| \cdot (d+1)$ (voir la ligne 1 de l'algorithme 9) où N_i est le nombre d'observations du nœud i et $(d+1)$ est le nombre de propriétés du *data-set* plus l'information de classe effective des observations. Dans ce cas, nous chargeons toutes les propriétés des observations du nœud en mémoire. Ce faisant, le sous-arbre allant du nœud courant jusqu'aux nœuds feuilles peut être construit sans mouvements de données supplémentaires. En effet, en supposant

que les nœuds enfants résultants de la division du nœud i sont impures et nécessitent une division, un sous-ensemble aléatoire de propriétés à tester pour chacun des deux nœuds est formé. Ce sous-ensemble peut contenir des propriétés qui n'appartiennent pas au sous-ensemble F du nœud père. Le fait de charger toutes les propriétés du *data-set* au niveau du nœud père, si l'espace mémoire disponible le permet, constitue donc, un pré-chargement des données susceptibles d'être demandées à la construction du sous-arbre complet. Ce pré-chargement est efficace si la construction se fait en profondeur, c'est-à-dire suivant la stratégie DFS (voir la définition donnée en section 2.4.4), car le nœud suivant à être traité après i est le nœud fils gauche, dont les observations constituent un sous-ensemble de l'ensemble des observations du nœud i.

Scénario 2, division complète d'un nœud

Dans ce scénario, l'espace mémoire ne peut pas contenir toutes les propriétés des observations du nœud i, mais peut, en revanche, contenir les propriétés qui appartiennent au sous-ensemble des propriétés à tester F. Cette condition est formulée comme suit : $|N_i| \cdot (|F|+1) \leq M < |N_i| \cdot (d+1)$ (voir la ligne 4 de l'algorithme 9), tel que $|N_i|$ est le nombre d'observations affectées au nœud, $(|F|+1)$ est la taille qu'occupe les propriétés à tester d'une observation et M est l'espace mémoire disponible. Dans ce cas, les propriétés à tester et l'information de classe des observations affectées à ce nœud sont chargées en mémoire et gardées jusqu'à la fin du traitement du nœud i. Ainsi, le traitement de i se fait sans mouvements de données supplémentaires. Ici, le volume des E/S est réduit par le chargement des données utiles uniquement.

Il est à noter que dans l'algorithme 9, les données sont chargées juste après la détection du scénario dans le cas des scénarios 1 et 2. Le cas du scénario 3 est détaillé ci-après.

Scénario 3, tentative de division par parties

Ce scénario se produit dans le cas où le volume des données utiles ne peut pas être contenu en mémoire. Cela se traduit par la condition suivante : $M < N_i \cdot (|F|+1)$ qui signifie que l'espace mémoire disponible ne suffit pas à contenir le sous-ensemble des propriétés à tester pour la division. L'idée est alors d'effectuer la tentative de division par parties pouvant être contenues en mémoire. Autrement dit, nous proposons d'effectuer la tentative de division sur chaque partie ce qui revient à décomposer la tentative de division de nœud selon les étapes qui vont suivre, résumées dans l'algorithme 10 et illustrées par la figure 4.6.

1. **Charger une partie des observations** du nœud i qui peut être contenue en

mémoire. Étant donné un espace mémoire de M, la taille d'une partie en nombre d'observations qu'elle contient est de $\frac{M}{|F|+1}$ tel que $(|F|+1)$ est le volume qu'occupe les données utiles d'une observation donnée. Dans la figure 4.6, la première partie des observations à être chargée contient : $\{A, B, C\}$, et la deuxième contient : $\{D, E, F\}$. La première étape consiste à charger les propriétés à tester des observations appartenant à chacune des parties (voir la ligne 2 de l'algorithme 10). Cela est réalisé en utilisant l'index construit par le module de chargement des données pour localiser les observations du *data-set*.

2. **Effectuer la tentative de division indépendamment** pour chaque partie, le module de construction des arbres de décision construit les $|F|$ arbres potentiels en parcourant les observations de la partie chargée en les affectant aux nœuds enfants selon chaque propriété de $|F|$ (voir la ligne 3 de l'algorithme 10). Comme montré par la figure 4.6, les observations de la partie 1 sont distribuées selon les propriétés f_1 et f_2.
3. **Répéter les étapes (1) et (2)** jusqu'à ce que toutes les parties soient traitées.
4. **Combiner les arbres potentiels obtenus.** L'objectif de cette étape est de fusionner les $|F|$ arbres potentiels (ligne 6 algorithme 10). La combinaison des arbres potentiels se fait de cette façon :
   (a) Grouper les nœuds enfants qui correspondent aux mêmes valeurs de propriétés. Dans la figure 4.6, on observe que pour chaque propriété (f_1 et f_2), les nœuds gauches (resp. droits) de chaque arbre potentiel sont combinés selon chaque propriété de F pour obtenir les $|F|$ arbres potentiels.
   (b) Compter le nombre d'observations par classe dans chaque nœud : le nombre d'observations par nœud est la somme des nombres d'observations par nœud sur toutes les parties. Ce comptage ne cause pas d'E/S supplémentaires puisqu'il est effectué au fur et à mesure de la tentative de division sur les parties.

   Le choix de la meilleure propriété pour la division est effectué de la même manière que dans la méthode traditionnelle.

La division par parties détaillée ci-dessus n'altère pas les arbres potentiels obtenus, car la mesure de pureté des nœuds enfants se fait après combinaison des arbres potentiels.

4.2.4 Nouvelle méthode de construction des RF

L'algorithme 11 synthétise la construction d'une RF selon la méthode RaFIO. Cette dernière repose sur les deux optimisations détaillées précédemment. La première optimi-

Algorithm 10: Traitement d'un nœud par parties

Data: D : Fichier *Data-set*, F : Ensemble des propriétés de division
Result: $|F|$ arbres potentiels

for $c \leftarrow 0$ **to** $\frac{|N_i| \cdot S(|F|+1)}{M}$ *-1* **do**
    Charger à partir de D la partie c
    // **Traiter chaque partie**
    for $f \leftarrow 0$ **to** $|F|-1$ **do**
        Tenter de diviser c selon la propriété f
Combiner les arbres potentiels obtenus de chaque partie

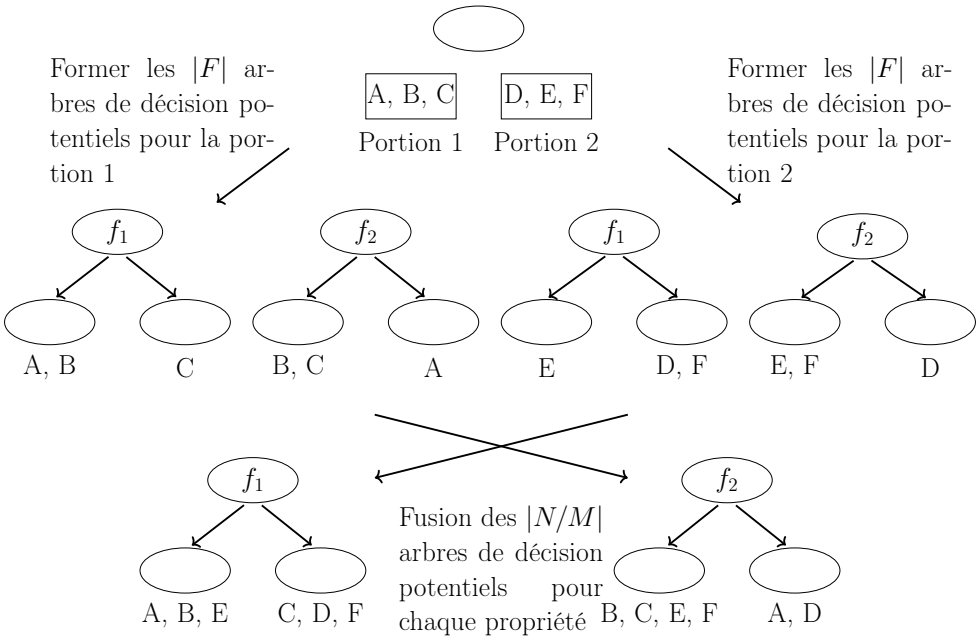


FIGURE 4.6 – Étapes de la tentative de division par parties

Algorithm 11: Algorithme RaFIO

```
   Data: Data-set
   Result: Forêt aléatoire
1  Construire l'arbre T_0 et réorganiser le Data-set (Algorithme 8)
2  for t ← 0 to T − 1 do
3      Créer un bootstrap
4      while il existe un nœud impure parmi les enfants du nœud courant do
5          Créer le sous-ensemble F de propriétés à tester
6          Définir le scénario et charger éventuellement les données (Algorithme 9)
7          if Scénario ≠ scénario 3 then
8              Effectuer une tentative de division du nœud avec la méthode classique
9          else
10             Effectuer la tentative de division du nœud par parties (Algorithme 10)
11         Choisir la propriété qui donne les nœuds les plus pures possibles
12         Diviser le nœud n selon la meilleure propriété
13     Ajouter l'arbre construit à la forêt aléatoire RF
14 retourner la RF
```

sation consiste à réorganiser le *data-set* selon l'algorithme 8 (ligne 1 de l'algorithme 11). Une fois le nouveau *data-set* créé, les arbres de décision de la forêt aléatoire sont construits selon l'algorithme 10 de sorte à ne pas charger les données inutiles et en tenant compte de l'espace mémoire disponible pour adapter la méthode de chargement (lignes 3-12 de l'algorithme 11).

4.3 Évaluation expérimentale

Dans cette section, nous évaluons la méthode RaFIO. Nous commençons par décrire la méthodologie d'évaluation utilisée. Par la suite, nous analysons les résultats obtenus.

4.3.1 Méthodologie expérimentale

Description des expériences

Notre objectif dans les expériences qui suivent est de mesurer l'effet des optimisations proposées dans RaFIO sur le temps total de construction d'une forêt aléatoire. Dans notre cas, la baisse du temps de construction de la RF traduit la réduction des E/S. Ces expériences sont organisées de manière à évaluer la performance générale de la méthode proposée puis, les spécificités de chaque optimisation.

— **Expérience 1 :** dans cette expérience, nous mesurons la performance générale de

RaFIO en la comparant à la méthode de construction d'arbre de décision disponible sur le *Framework Ranger*. Nous montrons aussi l'impact de chacune des optimisations sur la performance globale de RaFIO. Cette expérience est réalisée sous différentes contraintes mémoire N/M. Les valeurs N/M utilisées sont : $\{1, 2, 4, 8\}$ et le nombre d'arbres construits est de 25 (le nombre, par défaut, d'arbres construits dans une RF est de 100, nous choisissons d'utiliser 25 arbres dans cette expérience. Il sera montré dans l'expérience 2 que la réduction du temps d'exécution est quasiment stable lorsque le nombre d'arbres varie). Nous comparons également l'accès aux données à la demande, à la méthode de construction hybride BFS-DFS. En effet, puisque l'accès à la demande se base sur le fait que nous supposons que l'arbre est construit en DFS, il nous paraît important de confronter cette optimisation à une méthode qui emploie les deux stratégies de construction.

— **Expérience 2 :** l'objectif de cette expérience est d'analyser la réduction de temps d'exécution réalisée par la première optimisation, c'est-à-dire la réorganisation du *data-set*, lorsque l'on augmente le nombre d'arbres construits. Pour ce faire, nous fixons la contrainte mémoire à $N/M = 4$ et faisons varier le nombre d'arbres à construire T tel que $T = \{25, 50, 100, 150\}$.

— **Expérience 3 :** dans cette expérience, nous évaluons l'efficacité de l'accès à la demande aux données selon chacuns des scénarios de mémoire disponible par rapport au volume de données à traiter par nœud. Nous évaluons, pour chacun des scénarios, la réduction du temps d'exécution par la méthode proposée. Pour obtenir cette évaluation, nous procédons selon les étapes suivantes :

  — Pour RaFIO :

    1. Déterminer, pour chaque nœud, le scénario selon lequel il est traité ;
    2. Cumuler le temps d'exécution selon chaque scénario. On obtient alors le temps total de traitement des nœuds concernés par les scénarios respectifs 1, 2 et 3 ;
    3. Calculer la moyenne de temps d'exécution pour chacun des scénarios, obtenu en divisant le temps d'exécution du scénario par le nombre de nœuds concernés.

  — Pour la méthode de référence :

    1. Déterminer, pour chaque nœud, le scénario selon lequel il aurait été traité dans le cas de RaFIO ;
    2. Cumuler le temps d'exécution selon chaque scénario ;
    3. Calculer la moyenne de temps d'exécution pour chacun des scénarios.

Data-set	Nombre de propriétés	Nombre d'observations	Taille du fichier *Data-set* (Mo)
Covertype	54	581012	239.36
Wearable	8	75128	4.58
Adult	14	48842	5.21
Ecoli	8	336	0.02
Heart Failure	299	13	0.03
Wine	178	13	0.01
Poker	11	25010	2.1
Synthétique	16	100000	48.82
	32	5000	
	64	2500	
	182	1250	

TABLE 4.6 – *Data-sets* utilisés

Nous comparons les deux moyennes obtenues pour évaluer la réduction du temps d'exécution réalisée par chaque méthode. Les contraintes mémoire définies pour cette expérience sont les suivantes : $N/M = \{1, 2, 4, 8, 16\}$.

— **Expérience 4 :** dans cette expérience, l'objectif est d'évaluer l'efficacité de l'accès à la demande aux données par rapport à la méthode de référence lorsque l'on varie le nombre de propriétés du *data-set*. Le fait d'avoir plus de propriétés dans le *data-set* augmente le nombre de parcours sur les données nécessaire pour la division d'un nœud, d'où l'importance de mesurer l'impact de la baisse du volume de données inutiles lorsque le nombre de propriétés varie. Nous effectuons le test avec $d = \{16, 32, 64, 128\}$ et avec les contraintes mémoire suivantes : $N/M = \{1, 2, 4, 8\}$.

Métriques mesurées

La métrique utilisée pour les expériences précédentes est le pourcentage de réduction de temps d'exécution. Nous mesurons pour chaque expérience, le temps de construction de la forêt, le temps passé à faire des E/S et le temps de construction de l'arbre T_0 et de réécriture du *data-set* dans le cas de la première optimisation. Chaque mesure est répétée 5 fois. Les résultats donnés dans ce qui suit représentent les réductions de temps d'exécution moyens obtenus à partir des 5 mesures. Les réductions minimales et maximales sont également montrées. Dans cette évaluation, nous ne mesurons pas la qualité des résultats obtenus car les modifications apportées à l'algorithme n'ont pas d'effets sur les arbres obtenus.

Data-sets utilisés

Les *data-sets* utilisés dans nos expérimentations sont listés dans la table 4.6. Nous avons utilisé 3 *data-sets* réels (Covertype, Wearable, Adult) disponibles sur le dépôt de *data-sets* pour le ML UCI [41] et 4 *data-sets* synthétiques générés en utilisant le *Framework Scikit-Learn* [105]. Les *data-sets* Ecoli, Heart Failure, Wine et Poker ont été utilisés pour la validation de la propriété de similarité.

Plateforme de test

Les expériences de ces évaluations ont été effectuées sur une machine virtuelle Linux configurée avec 4 Go de mémoire pour se rapprocher des volumes mémoire disponibles sur certaines plateformes embarquées (voir la table 1.1 page 22). Les contraintes mémoires ont été appliquées en utilisant le mécanisme de *cgroup* [88] qui permet, entre autres, de limiter le volume de mémoire physique qui peut être alloué à un processus. La taille d'un bloc d'E/S est fixée selon la taille de page par défaut de Linux qui est de 4 Ko.

4.3.2 Résultats et discussion

Dans cette section, nous montrons les résultats des expériences d'évaluation et nous les analysons.

Expérience 1 : Évaluation de la méthode RaFIO

Évaluation de la réorganisation du *data-set* : la figure 4.7 montre la réduction réalisée par la réorganisation des observations du *data-set* de manière à augmenter la localité spatiale. On relève les observations suivantes :

— La réduction du temps d'exécution varie entre 24 et 72% avec une moyenne de 58% de réduction.

— Globalement, la réduction du temps d'exécution diminue à mesure que l'on augmente la contrainte mémoire N/M. En effet, bien que le *data-set* est organisé de sorte à augmenter la localité spatiale, lorsque l'on augmente la contrainte mémoire, un nœud est plus susceptible de contenir plus d'observations que peut en contenir la mémoire. Par conséquent, la tentative de division de ce nœud engendre plus de mouvements de blocs entre la mémoire principale et le stockage secondaire, ce qui rend cette optimisation moins performante.

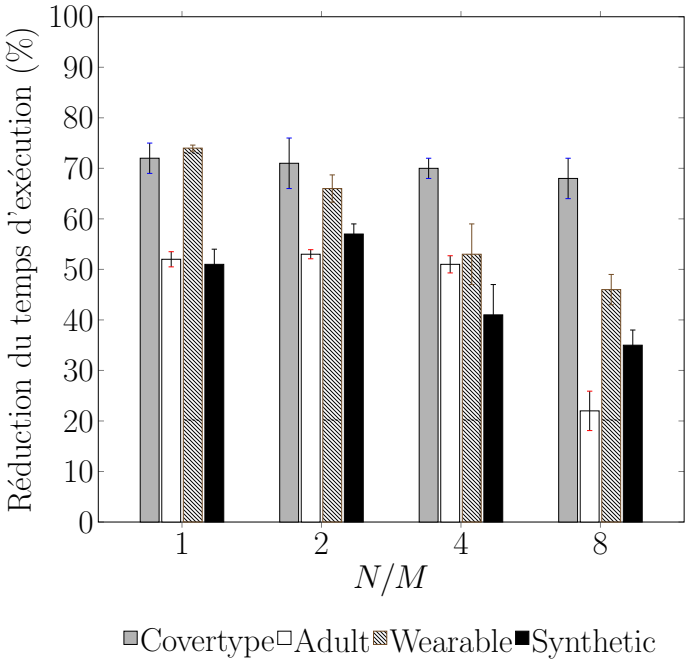


FIGURE 4.7 – Réduction du temps d'exécution par la réorganisation du *data-set*

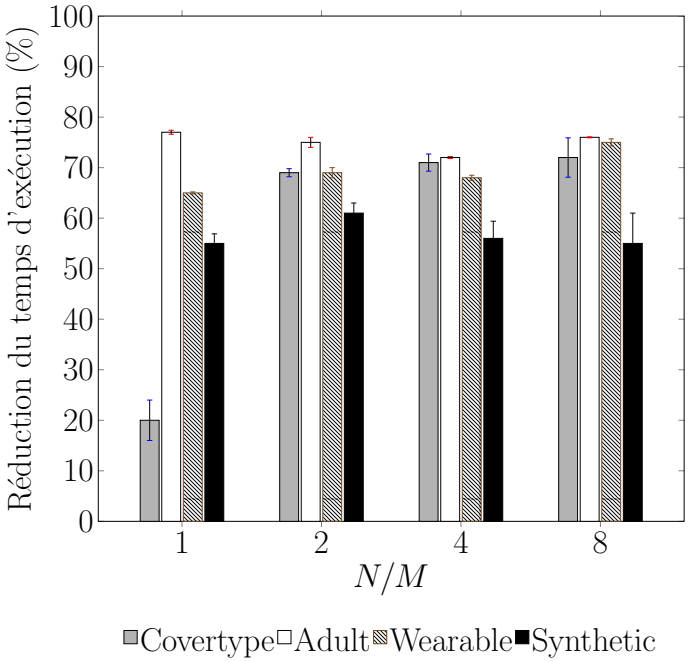


FIGURE 4.8 – Réduction du temps d'exécution par l'accès aux données à la demande

Évaluation de l'accès à la demande des données : la figure 4.8 montre la réduction du temps d'exécution par rapport à *Ranger* obtenue en effectuant des accès aux données à la demande lors de la construction de l'arbre de décision. Nous en tirons les observations suivantes :

— La réduction moyenne du temps de construction d'une forêt aléatoire est de 64%.
— Globalement, plus la contrainte mémoire est élevée, plus la réduction du temps de construction est importante. Cela est dû au fait que l'accès aux données à la demande réduit des E/S en ne chargeant que les observations affectées à un nœud donné. Par conséquent, le temps d'E/S est bas en comparaison à la méthode de référence *Ranger*.

Évaluation de l'accès à la demande des données par rapport à la méthode BFS-DFS [5] : la figure 4.9 montre les réductions du temps d'exécution par l'accès aux données à la demande par rapport à la méthode de construction hybride BFS-DFS [5]. Cette dernière consiste à appliquer une méthode de construction en largeur (BFS) sur les premiers niveaux de l'arbre, puis une méthode en profondeur (DFS) sur les niveaux restants. On observe que pour le *data-set Covertype*, la méthode BFS-DFS est plus efficace que la méthode proposée lorsque $N/M = 1$. Pour le reste des mesures, on observe que la méthode proposée réduit le temps de construction d'un arbre de décision de 15 à 69%. Cela est légèrement plus faible que la réduction réalisée par notre méthode par rapport au *framework Ranger*.

Bien que la méthode BFS-DFS augmente la localité spatiale, le problème des mouvements entre la mémoire principale et l'espace de *swap* subsiste lorsque la contrainte mémoire N/M est forte. Par conséquent, la méthode d'accès aux données à la demande, qui prend en considération l'espace mémoire disponible pour construire l'arbre de décision, a de meilleures performances que la méthode BFS-DFS.

Évaluation de la combinaison des deux optimisations : la figure 4.10 montre la réduction du temps d'exécution réalisée en combinant les deux optimisations. Nous en tirons les observations suivantes :

— La réduction du temps d'exécution sur l'ensemble des mesures est de 78%.
— En comparant les réductions moyennes de temps d'exécution entre les figures 4.7, 4.8 et 4.10, on obtient la réduction moyenne la plus élevée. Cela est dû au fait que seules les observations utiles sont chargées en mémoire, combiné au fait que les blocs de données présentent une plus forte localité spatiale.

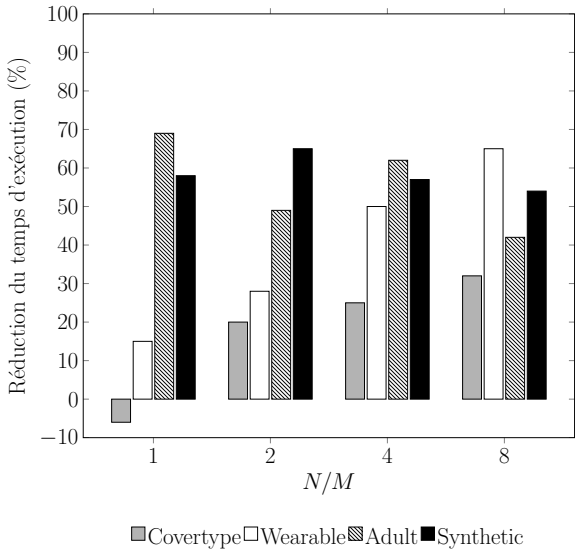


FIGURE 4.9 – Réduction du temps d'exécution réalisée par l'accès à la demande aux données par rapport à la méthode BFS-DFS

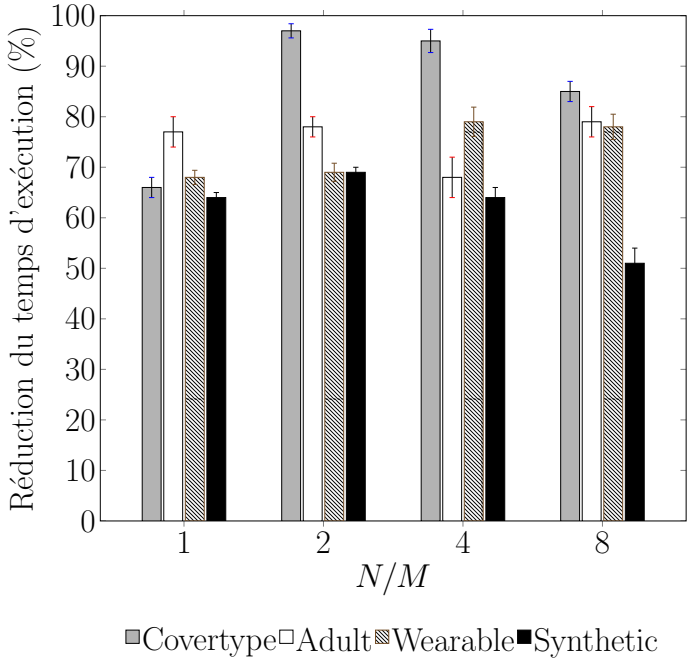


FIGURE 4.10 – Réduction du temps d'exécution par la combinaison des deux optimisations

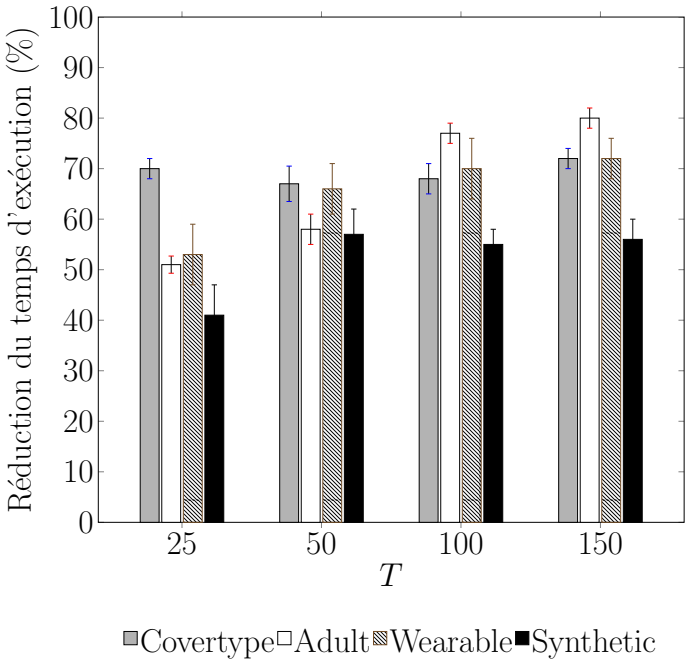


FIGURE 4.11 – Réduction du temps d'exécution par la réorganisation du *data-set* et en faisant varier T

Synthèse de l'expérience 1 : les expériences réalisées montrent que la réorganisation des données, qui a pour but d'augmenter la localité spatiale, réalise une réduction moyenne du temps de construction de 58%, et avec une tendance décroissante lorsque la contrainte mémoire augmente. L'accès aux données à la demande réduit le temps de construction d'une *RF* de 64% en moyenne avec une tendance croissante avec l'augmentation de la contrainte mémoire. En résumé, cette première expérience montre que RaFIO réduit le temps de construction d'une forêt aléatoire de 75% en moyenne par rapport au *Framework Ranger*.

Expérience 2 : évaluation de la réorganisation des données pour un nombre variable d'arbres construits

Réductions du temps d'exécution pour des nombres variables d'arbres : la figure 4.11 montre la réduction moyenne du temps de construction de forêts aléatoires constituées d'un nombre T d'arbres différents, en appliquant la réorganisation du *data-set*. On note les observations suivantes :

— La réduction du temps d'exécution réalisée pour différents *data-sets* varie entre 41 et 80%.

— Pour un *data-set* donné et pour un espace mémoire disponible, la réduction du temps d'exécution réalisée varie peu avec l'augmentation du nombre d'arbres

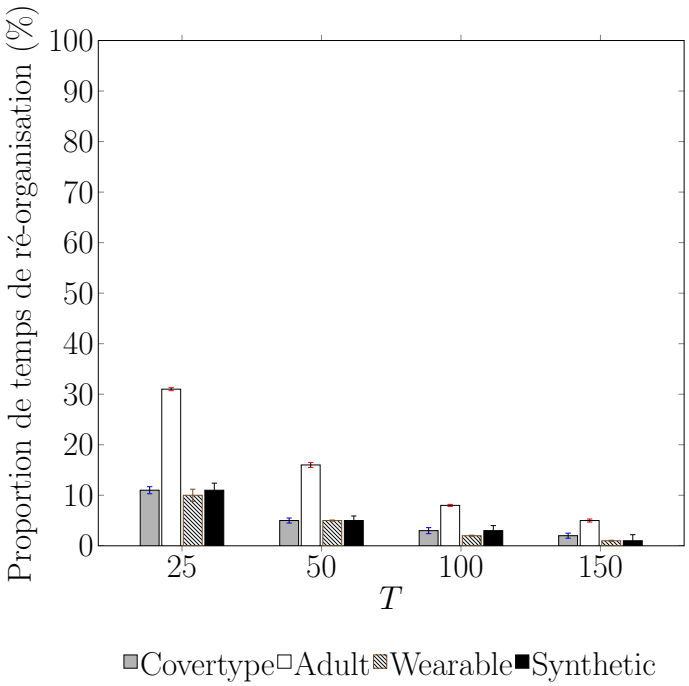


FIGURE 4.12 – Réduction du temps d'exécution par la réorganisation du *data-set* et en faisant varier T

construits. Pour le *data-set Covertype* la réduction du temps d'exécution reste aux environs de 70% quelle que soit la valeur de T. Pour le *data-set* synthétique, la réduction de temps d'exécution se stabilise aux alentours de 56%. Pour le *data-set Adult* , et dans une moindre mesure, le *data-set Wearable*, les réductions de temps d'exécution augmentent avec l'augmentation du nombre d'arbres T. Nous en détaillons les raisons dans l'évaluation qui suit.

Évaluation de la proportion de temps passée à réorganiser les données : les résultats de cette évaluation sont montrés dans la figure 4.12. On observe ce qui suit :

— Pour le *data-set Adult*, la proportion de temps consacré à la construction de l'arbre T_0 et la réorganisation du *data-set* est importante lorsque le nombre d'arbres est bas $T = 25$. Ce coût est amorti à mesure que l'on augmente le nombre d'arbres construits.

— Pour les autres *data-sets*, la proportion mesurée pour $T = 25$ est relativement faible. Par conséquent, malgré l'amortissement de ce coût lorsque le nombre d'arbres augmente, cela n'a pas d'impact visible sur l'évolution de la réduction du temps d'exécution avec l'augmentation du nombre d'arbres construits (voir la figure 4.11).

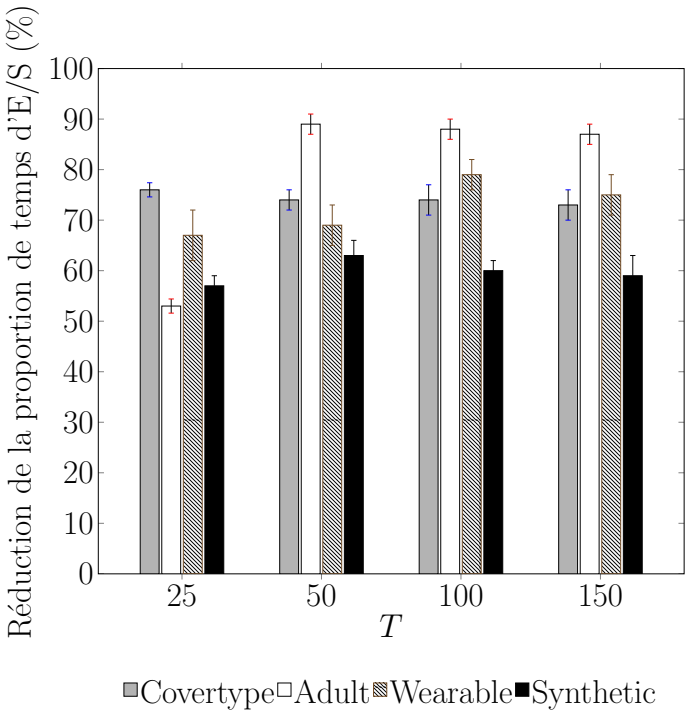


FIGURE 4.13 – Réduction du temps d'exécution par la réorganisation du *data-set* et en faisant varier T

Évaluation de la proportion de temps d'E/S par rapport au temps de construction d'une RF : la figure 4.13 montre la réduction de la proportion du temps d'E/S obtenue grâce à la réorganisation du *data-set*. On observe que cette réduction suit la courbe de réduction du temps d'exécution donnée dans la figure 4.11. En effet, l'augmentation de la localité spatiale permet d'accéder à moins de blocs d'E/S. Par conséquent, le temps consacré aux E/S est plus bas que dans la méthode de référence, et par conséquent, le temps de construction d'une *RF* s'en trouve réduit.

Synthèse de l'expérience 2 : l'objectif de cette expérience est d'observer le pourcentage de temps de réorganisation du *data-set* par rapport au temps de construction de la forêt aléatoire et d'estimer l'impact de cette réorganisation sur le temps de construction total. Pour cela, nous avons construit des forêts de différents nombres d'arbres. L'observation qui en découle est que le pourcentage de temps consacré à la réorganisation des données est faible (8% en moyenne) et est amorti à mesure que le nombre d'arbres de la forêt augmente.

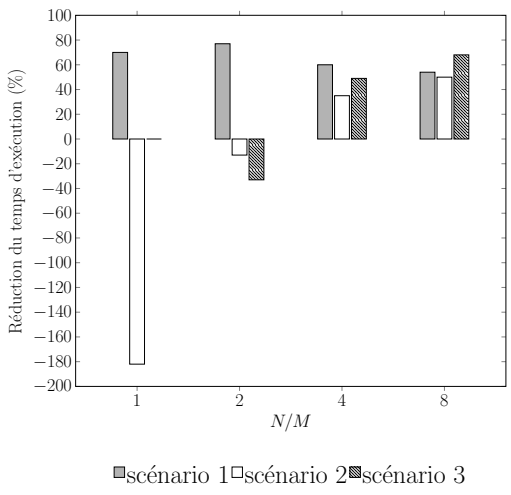


FIGURE 4.14 – Non pondérée

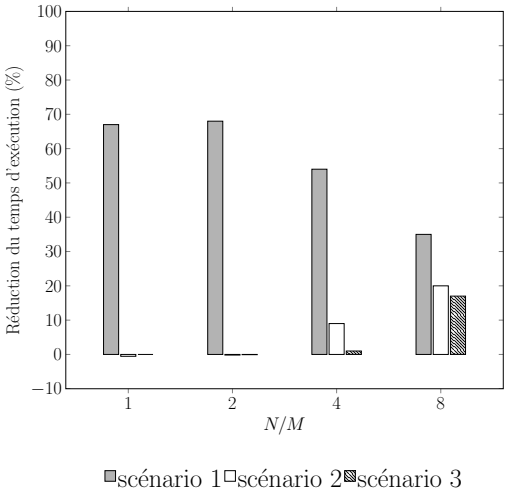


FIGURE 4.15 – Pondérée

Réduction du temps d'exécution avec chaque scénario

Expérience 3 : évaluation de la performance de chaque scénario d'accès à la demande aux données

La figure 4.14 montre les réductions de temps d'exécution obtenues grâce à l'accès aux données à la demande avec chacun des trois scénarios par rapport à la méthode *Ranger*. La figure 4.15 montre les réductions des temps d'exécution pondérées obtenues avec chacun des trois scénarios. Ces valeurs pondérées sont obtenues en multipliant les réductions données dans la figure 4.14 par les proportions de nombre du nœuds concernés par le scénario. On en tire les observations suivantes :

— Pour $N/M = 1$, puisque l'espace mémoire est suffisant pour contenir le *data-set*, le scénario 3 ne se produit jamais. Le scénario 2 se produit pour les nœuds proches de la racine de l'arbre, car un nombre important d'observations leurs sont affectés alors que l'espace mémoire contient d'autres structures. On observe, à partir de la figure 4.15, que la stratégie proposée pour ce scénario est moins efficace que la méthode de référence. Cela est dû au fait que pour effectuer un chargement à la demande, l'index qui permet de localiser les données à charger est nécessaire et utilisé de manière intensive, s'ajoute à cela le fait que la majorité des observations soient chargées, car le nombre d'observations affectées à un nœud sont nombreuses. Par conséquent, cette stratégie augmente le coût en E/S par rapport à la stratégie de référence. Néanmoins, comme les nœuds concernés par ce scénario sont peu nombreux, ce coût supplémentaire est négligeable par rapport au gain réalisé avec le scénario 1 telle que montrée la figure 4.15. La stratégie globale d'accès à la demande est, par conséquent, performante par rapport à la méthode de référence tel

que montré la figure 4.8.

— Pour $N/M = 2$, on observe que les stratégies employées pour les scénarios 2 et 3 est moins efficace que la méthode de référence. La raison en est la même que dans le cas du scénario 2 avec $N/M = 1$; le volume des E/S dû aux chargements des parties de données dépasse le volume des E/S dû aux mouvements des blocs entre la mémoire principale et l'espace de stockage secondaire puisque le volume mémoire disponible est proche du volume de données à traiter. Néanmoins, puisque peu de nœuds sont concernés par ces scénarios, la stratégie globale reste performante par rapport à la méthode de référence.

— Pour $N/M = 8$, on observe que les trois scénarios réalisent les réductions respectives de 35, 20 et 17% par rapport à la méthode de référence.

En somme, la stratégie associée au scénario 1 est efficace lorsque la contrainte mémoire est faible (N/M petit), alors que les scénarios 2 et 3 sont efficaces pour de fortes contraintes mémoire. L'utilité des scénarios 2 et 3 est également liée à la profondeur de l'arbre. Ainsi, plus l'arbre est profond dans un contexte de forte contrainte mémoire, plus les stratégies liées aux scénario 2 et 3 sont performantes.

Expérience 4 : Évaluation de la performance de l'accès à la demande en faisant varier le nombre de propriétés *d*

La figure 4.16 montre les réductions de temps d'exécution réalisées par l'accès aux données à la demande, par rapport à la méthode de construction des *RF* telle que mise en œuvre dans le *Framework Ranger* et ce, pour différents nombres de propriétés de *data-set* *d*. En résumé, la réduction du temps d'exécution varie entre 15 et 90%. Deux observations peuvent être tirées de cette expérience :

— La méthode proposée est plus performante avec les nombres de propriétés faibles. Cela est dû à notre méthodologie d'expérimentation. En effet, dans un souci de conservation de la même contrainte mémoire (N/M), nous avons fixé le volume du *data-set* et fait varier le nombre de propriétés. Par conséquent, en augmentant le nombre de propriétés tout en gardant le même volume de données, le nombre d'observations par *data-set* diminue, ce qui réduit les profondeurs des arbres construits. Comme le nombre d'observations du *data-set* est plus petit, le nombre de nœuds d'un arbre est aussi plus petit. Or, les E/S s'opèrent à la division des nœuds ; c'est pour cela que la méthode semble plus rentable avec les nombres de propriétés faibles.

— Pour un même nombre de propriétés, la réduction de temps d'exécution diminue entre deux valeurs successives de N/M. Cela se produit pour $d = 16$ entre $N/M =$

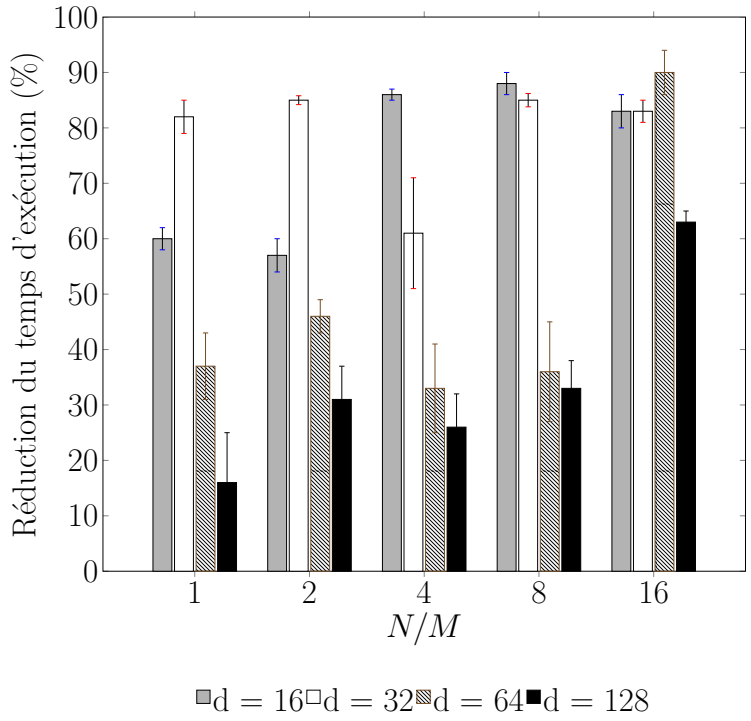


FIGURE 4.16 – Réduction du temps d'exécution réalisée par la réorganisation du *data-set* et en faisant varier d

1 et $N/M = 2$, et pour $d = 32$, $d = 64$ et $d = 128$ entre $N/M = 2$ et $N/M = 4$. Cela est dû au fait que plus les valeurs de N/M sont élevées, plus le scénario 3 est susceptible de se produire, ce qui implique davantage d'opérations d'E/S que dans les scénarios 1 et 2.

4.4 Conclusion

Dans ce chapitre, nous avons présenté une méthode qui réduit le volume des E/S lors de la construction d'une forêt aléatoire (*Random Forest I/O-aware algorithm*, RaFIO). En effet, nous avons montré que lors de la construction d'une forêt aléatoire dans un environnement contraint en mémoire, une proportion importante du temps est consacrée aux mouvements de données entre la mémoire principale et le stockage secondaire.

Afin de réduire le coût dû aux E/S, la méthode que l'on propose est structurée autour de deux principes : **(1)** la réorganisation du *data-set* : l'objectif est de regrouper les observations du *data-set* susceptibles d'être accédées dans une même fenêtre temporelle sur les mêmes blocs de données, de sorte à augmenter la localité spatiale. Les « groupes » d'observations susceptibles d'être accédées dans une même fenêtre temporelle sont déduits à partir de la construction d'un premier arbre qui met en évidence les observations affectées

aux mêmes feuilles. **(2)** l'accès aux données à la demande : l'objectif de cette méthode est de lever l'hypothèse selon laquelle le *data-set* est entièrement chargé en mémoire qui est difficilement réalisable lorsque l'espace mémoire est restreint. Dans la méthode proposée, les données sont chargées au moment de la formation des nœuds de l'arbre de sorte à ne charger que les observations qui seront effectivement utilisées.

Nos expériences montrent que RaFIO réduit le volume des E/S au moment de la construction d'une forêt aléatoire et par suite, le temps de construction. La réorganisation du *data-set* réduit le temps de construction de 58% en moyenne. L'accès aux données à la demande se traduit par une réduction moyenne de 64% et la combinaison des deux optimisations atteint les 75% de réduction moyenne.

Le prochain chapitre synthétise les travaux menés dans cette thèse, et dresse quelques perspectives de travaux futures.

Chapitre 5

Conclusion et perspectives

Cette thèse s'est intéressée à la contrainte mémoire sur les plateformes embarquées à laquelle fait face l'émergence de l'EDGE INTELLIGENCE (EI). Le présent chapitre clôt ce manuscrit en rappelant le contexte de cette thèse, ainsi que nos contributions, puis dresse les perspectives de travaux futurs.

Sommaire

5.1 Conclusion

Dans cette première section, nous rappelons le contexte et la problématique adressée. Par la suite, nous résumons nos contributions et les principaux résultats obtenus.

5.1.1 Contexte de la thèse

Avec la démocratisation des divers dispositifs de collecte de données tels que les *smartphones*, les drones, les véhicules intelligents, les accessoires de domotique, le volume de données mondial croit de manière exponentielle. L'extraction efficace de connaissance est,

alors, un enjeu important afin de tirer profit de ces données. Cela passe par la mise en place d'architectures et d'algorithmes performants.

Dans ce contexte, l'EDGE INTELLIGENCE (EI) est un paradigme émergent qui consiste à extraire de la connaissance au plus près des dispositifs de collecte de données, au lieu de construire des modèles intelligents sur le *cloud*. L'objectif est de réduire le coût des communications, qui représente 70% de la consommation énergétique des dispositifs embarqués [18], et d'assurer la confidentialité des données. Concrètement, cela consiste à exécuter localement les algorithmes de *Machine Learning* (ML), ou une partie de ces algorithmes sur les dispositifs de collecte ou à leur périphérie.

Néanmoins, l'EI soulève plusieurs défis scientifiques, entre autres, la lenteur d'exécution des algorithmes de ML [161] et leurs exigences en terme d'espace mémoire [18]. En effet, les dispositifs embarqués présentent des capacités mémoire limitées et ne sont pas conçus pour une exécution en continue d'une tâche. Par corollaire, l'adaptation des algorithmes de ML pour permettre leur exécution dans des environnements limités en ressources est un champ de recherche en exploration.

5.1.2 Problématique adressée

Dans cette thèse, nous nous sommes particulièrement intéressée à la problématique des mouvements de données entre les niveaux de la hiérarchie mémoire pendant la phase d'apprentissage des algorithmes de ML. En effet, la limitation de l'espace mémoire des dispositifs embarqués par rapport aux volumes de données à traiter rend les mouvements de données, entre la mémoire principale et le stockage secondaire, inévitables. De manière générale, la mémoire principale et le stockage secondaire sont respectivement basés sur les technologies DRAM et *Flash*. Or, le rapport de temps d'accès aux données entre ces deux technologies est de l'ordre de 2500 [18]. Par conséquent, ces mouvements ralentissent considérablement la phase d'apprentissage. La problématique posée pour cette thèse est formulée comme suit : **comment réduire le volume des E/S des algorithmes de ML, pour permettre leur intégration dans les plateforme embarquées ?**

Les algorithmes de ML sont très hétérogènes, autrement dit, les causes algorithmiques des E/S sont différentes d'un algorithme à l'autre. Par conséquent, il est difficile de proposer une méthode de réduction adaptée à l'ensemble des algorithmes. Nous avons, donc, procédé par cas d'étude d'algorithmes de ML. Nous avons choisi, pour cette thèse, l'algorithme de *clustering* *K*-MEANS et l'algorithme de classification RANDOM FORESTS.

5.1.3 Contributions

Pour les cas d'études choisis, nous avons adopté la démarche consistant à : **(1)** analyser le motif d'E/S de ces algorithmes ; **(2)** identifier les parties algorithmiques intensives en E/S ; **(3)** proposer une solution qui réduit le volume d'E/S.

Algorithme *K-means* économe en E/S, K-MLIO

La première contribution de cette thèse est la proposition de l'algorithme : *K-Means with a Low I/O* (K-MLIO). Cette proposition a été faite à l'issue du processus suivant :

1. Analyse du motif des E/S de l'algorithme *K-means* : nos analyses, théorique et expérimentale de l'algorithme *K-means*, ont montré qu'une large proportion du temps d'exécution est consacrée aux E/S. Pour un *data-set* 4 fois plus volumineux que l'espace mémoire disponible, 70% du temps d'apprentissage est dû aux E/S.
2. Identification des causes algorithmiques des E/S : nous avons mis en évidence le fait que le *data-set* soit parcouru autant de fois que le nombre d'itérations nécessaires à la convergence de l'algorithme. Dans le cas où l'espace mémoire est de taille inférieure au volume de données à traiter, plusieurs parcours complets du *data-set* causent de nombreux mouvements entre la mémoire principale et le stockage secondaire.
3. Proposition d'un nouvel algorithme : l'objectif de notre méthode est de décorréler le nombre de parcours du *data-set* du nombre d'itérations de l'algorithme. Pour cela, nous appliquons le principe « diviser pour régner ». Notre méthode consiste à appliquer l'algorithme *K-means* jusqu'à convergence, sur des parties du *data-set*, pouvant être contenues en mémoire. Ce faisant, chaque partie du *data-set* est chargée une seule fois en mémoire. Sur la base des résultats obtenus, l'agrégation se fait en formant un *data-set* pouvant être contenu en mémoire, et qui est représentatif de tous les *clusters* du *data-set* originel. Le *K-means* est ensuite appliqué sur ce *data-set* pour aboutir au résultat final.

L'évaluation de cette méthode a montré que le temps d'apprentissage est réduit de 60% en moyenne par rapport à la méthode *K-means* classique.

Algorithme de RF économe en E/S, RaFIO

Notre deuxième cas d'étude a été l'algorithme des RF. Pour cet algorithme, nous avons proposé la méthode *Random Forest I/O-aware algorithm* (RaFIO), qui est basée sur deux

mécanismes indépendants et complémentaires. La démarche suivie pour ce cas d'étude est la suivante :

1. Analyse du motif des E/S de l'algorithme de construction des RF : de même que pour le *K-means*, nous avons analysé, de manière théorique et expérimentale le motif des E/S, lors de la construction d'une RF. Nous avons montré que pour un *data-set* 4 fois plus volumineux que l'espace mémoire disponible, la proportion du temps d'E/S représente près de 90% du temps de construction d'un arbre de décision.
2. Identification des causes algorithmiques des E/S : nous en avons identifié deux : **(1)** le mouvement de certaines données inutiles lors de la construction d'un arbre de décision, car appartenant au *data-set* mais pas à l'ensemble d'apprentissage de l'arbre de décision (*bootstrap)*, **(2)** la faible localité spatiale des blocs de données du *data-set*, autrement dit, les données nécessaires au traitement d'un nœud de l'arbre de décision sont distribuées sur plusieurs blocs.
3. Proposition d'un nouvel algorithme : nous avons proposé une méthode de réduction des E/S qui puisse répondre aux problèmes identifiés. Elle est fondée sur les deux mécanismes suivants :
   (a) La réorganisation du *data-set* sur le périphérique de stockage : l'objectif de ce mécanisme est d'augmenter la localité spatiale des données. Il consiste à réécrire le *data-set* tel que les observations susceptibles d'être traitées dans une même fenêtre temporelle se retrouvent dans les mêmes blocs. Afin de déduire les observations susceptibles d'être accédées dans un intervalle de temps court, nous nous basons sur la conjecture selon laquelle une paire d'observations appartenant à un même nœud au niveau d'un arbre de décision a une forte probabilité d'appartenir au même nœud dans un autre arbre de la même forêt. Cette conjecture nous permet de déduire, à partir du premier arbre construit, les observations susceptibles d'être accédées dans un intervalle de temps court. Le *data-set* est alors réécrit selon cette répartition.
   (b) L'accès aux données à la demande : cette stratégie a pour objectif de ne charger en mémoire que les données qui seront effectivement utilisées. En d'autres termes, elle s'affranchit de l'hypothèse selon laquelle le *data-set* est entièrement contenu en mémoire. Pour mettre en œuvre cette stratégie, nous avons identifié les différentes configurations espace mémoire disponibles - volume de données à traiter, possibles lors de la division d'un nœud. Pour chacune des configurations, nous proposons une méthode de chargement de données et de division du nœud adaptée, de manière à utiliser les données présentes en mémoire le plus longtemps possible sans en charger de nouvelles.

L'évaluation de RaFIO a montré que la réorganisation du *data-set* réduit le temps de construction d'un RF de 70% en moyenne par rapport à la méthode classique. L'accès aux données à la demande, quant à elle, le réduit de 60%.

5.2 Perspectives

Dans cette section, nous présentons quelques perspectives de travaux futurs que nous classons selon leur terme.

5.2.1 Perspectives à court terme

Dans cette section, nous donnons les améliorations possibles des travaux présentés dans cette thèse.

Améliorations possibles de K-MLIO

Le principaux facteurs qui peuvent limiter la performance de la méthode K-MLIO sont la complexité des données et leur distribution sur le *data-set* à laquelle nous avons répondu par la méthode K-MLIOR, mais qui peut toujours être améliorée. Dans ce qui suit, nous décrivons quelques améliorations possibles de K-MLIO :

— Intégration des méthodes d'élagage de calcul au niveau du *K-means* sur les *chunks* : dans la méthode K-MLIO, le *K-means* est appliqué sur plusieurs *chunks* de données jusqu'à convergence. Dans le cas de *data-sets* à faible index de séparation (les données sont peu séparées), la convergence de l'algorithme *K-means* est plus lente. Tel que décrit dans la section 2.4.3 page 65, il existe plusieurs méthodes d'élagage [44, 62, 39, 37, 13] qui permettent d'éviter des calculs de distance, et donc, les accès aux données. L'application de telles méthodes permettrait de réduire les temps d'application du *K-means* sur les *chunks*. Néanmoins, les méthodes d'élagage de calculs de distances exigent le maintien de bornes inférieures et supérieures de distances de chaque observation, par rapport aux centres des *clusters*, élevant ainsi l'empreinte mémoire de l'algorithme. Il existe, alors, un compromis à faire entre la complexité du *data-set* à traiter (index de séparation), et le coût mémoire dû à l'utilisation des méthodes d'élagage, de sorte qu'il est plus intéressant d'appliquer les méthodes d'élagage lorsque la complexité du *data-set* est élevée. L'efficacité de K-MLIO peut être améliorée en intégrant les méthodes d'élagage qui peuvent être utilisées lorsque le *chunk* est complexe (ses données sont peu séparées). La détection de complexité d'un *chunk* peut se faire sur la base du nombre d'itérations effectuées,

autrement dit, au-delà d'un certain seuil d'itérations effectuées, à déterminer, l'emploi des méthodes d'élagage est intéressant.

— Détection de la distribution des données : dans cette thèse nous avons mis en évidence le fait qu'une distribution non uniforme des données sur le *data-set*, ralentit la phase de calcul des *K-means* partiels de la méthode K-MLIO. Une distribution non uniforme des données signifie que dans un *chunk*, la proportion du nombre d'observations d'un *cluster* présentes dans le *chunk* par rapport au nombre total d'observations du *cluster* est différente d'un *cluster* à l'autre. Nous avons répondu à cette problématique en proposant la méthode K-MLIOR qui consiste à effectuer une randomisation des blocs du *data-set*. Cependant, nous relevons deux défauts à cette solution : **(1)** il existe une probabilité très faible de retrouver une distribution similaire à la distribution originelle, **(2)** nous ne connaissons pas, a priori, la distribution des données dans le *data-set*, et donc, nous ne savons pas si la randomisation est nécessaire. Par conséquent, il serait intéressant de proposer une phase de pré-traitement du *data-set*, qui consiste à détecter la distribution des données de manière grossière. La connaissance de la distribution des données pourrait avoir plusieurs avantages : la formation de *chunks* partiels se ferait non pas par randomisation, mais en tenant compte de la distribution des données sur les *chunks* ; enfin, la connaissance de la distribution de données pourrait permettre de classifier les *chunks* par ordre de difficulté de traitement (*clustering*). Une telle information nous permet d'enrichir la perspective à moyen/long terme ≪Parallélisation des algorithmes de ML≫ décrite en section 5.2.2.

5.2.2 Perspectives à moyen/long terme

Dans cette section, nous présentons quelques pistes de travaux futurs à plus long terme, sur lesquelles ouvre cette thèse.

Gestion de la consommation énergétique lors de l'exécution des application de ML

Au vu des contraintes énergétiques qui existent sur les dispositifs *Edge*, il serait particulièrement intéressant de prendre en considération les propriétés architecturales et énergétiques des plateformes qui exécutent ces algorithmes, afin de réaliser un compromis énergie-temps satisfaisant. La modulation de fréquence des processeurs est parmi les méthodes répandues de réduction de consommation énergétique [12, 11]. Dans [132], il a été démontré que dans le cas d'une application intensive en accès à la hiérarchie mémoire, il est intéressant de réduire la fréquence d'un processeur de sorte à réduire sa consomma-

tion énergétique sans ralentir l'exécution de l'application. Dans le cas des algorithmes K-MLIO et RaFIO, les phases intensives en calculs et en E/S s'alternent. Il serait, donc, intéressant d'explorer la possibilité d'exploiter cette particularité des méthodes proposées afin de moduler la fréquence de sorte à favoriser la performance durant les phases intensives en calculs, et l'économie en énergie durant les phases intensives en E/S.

Parallélisation des algorithmes de ML

L'approche « diviser pour régner » utilisée dans K-MLIO et dans la gestion du scénario 3 d'accès à la demande de RaFIO se prête à la parallélisation, ce qui permettrait d'accélérer l'apprentissage. Nous distinguons deux cas de distributions : le premier cas se présente dans une architecture à mémoire partagée, qui correspond dans cadre de l'EI à une architecture centralisée, où l'apprentissage se fait entièrement sur une seule plateforme. Si cette dernière a une architecture multi-processeurs, la parallélisation peut se faire localement. Le deuxième cas de parallélisation se présente dans un système distribué, qui correspond à une architecture *Edge* décentralisée ou hybride, où plusieurs dispositifs collaborent pour apprendre un modèle (voir en section 2.1.2 page 31 la description des architectures de l'EI). Dans ce qui suit, nous décrivons les méthodes de parallélisation de nos solutions selon chacun des cas, ainsi que les problématiques qui peuvent s'y rapporter.

Cas d'une architecture à mémoire partagée. Dans le cas où l'apprentissage se fait sur une seule platforme, et que cette dernière a une architecture multi-processeurs hétérogènes (CPUs multi-cœurs, GPUs), un bon compromis performance-énergie peut être réalisé [52]. Dans ce contexte, *mapping* des tâches de manière à accélérer leurs exécutions admet plusieurs solutions de qualités variables selon le compromis énergie-temps réalisé. Cette problématique a été traitée dans [70] pour des réseaux de neurones convolutionnels. Les auteurs proposent d'effectuer le *mapping* sur une architecture multi-processeurs en utilisant un algorithme génétique. Plus généralement, le problème de *mapping* des tâches sur une architecture multi-cœurs/multi-processeurs a été bien exploré dans la littérature. Dans [51], les auteurs emploient des méthodes de ML afin d'estimer la performance des *mapping* de tâches pour choisir, par la suite, le plus performant. Ces techniques de l'état-de-l'art peuvent s'appliquer à K-MLIO et RaFIO. Dans ce qui suit, nous décrivons les particularités de cette problématique dans le cadre précis de ces deux méthodes :

— Dans le cas de K-MLIO, les *chunks* doivent être chargés de manière parallèle pour être traités par les différents cœurs/processeurs. La question qui peut alors se poser est de connaître le comportement de la parallélisation des E/S au niveau noyau et

son impact sur l'efficacité de K-MLIO. Dans le cas où la parallélisation des E/S n'est pas gérée, autrement dit, les chargements des *chunks* ne peuvent pas se faire de manière parallèle, il serait intéressant de proposer un mécanisme qui permettrait de palier ce problème.

— Dans le cas d'un système composé de plusieurs CPUs ou/et GPUs, la parallélisation des traitements de *chunks* devrait tenir compte de la bande passante de la mémoire principale entre les processeurs. Une solution possible serait de réaliser une phase de mesure et calibration avant le lancement de K-MLIO, pour prendre en compte l'architecture disponible (taille mémoire disponible, taille des caches, fréquences processeur utilisables) et fixer ensuite les paramètres de parallélisation de K-MLIO (taille des *chunks*, *mapping* des *chunks*, fréquence à utiliser sur chacun des cœurs/processeur) afin de mieux gérer le partage de la bande passante mémoire et le différentiel de performances entre les processeurs.

— Pour RaFIO : La parallélisation peut être envisagée au niveau des arbres, en affectant à chaque cœur/processeur, la construction d'un arbre. Le mécanisme d'accès aux données à la demande, charge pour chacun des arbres en cours de traitement les données du nœud à diviser. Or, au vu du caractère aléatoire des *bootstraps*, la probabilité que les nœuds en cours de traitement à un instant donné (sur différents cœurs) contiennent des observations communes est importante, notamment sur les premiers niveaux des arbres. Cela augmente l'espace inutilement occupé par des données redondantes. Par conséquent, une méthode d'indexation efficace des blocs déjà en mémoire permettrait d'éviter de charger les mêmes données plusieurs fois. De même, cette technique d'indexation permettrait de réduire les redondance de données propres aux constructions des arbres de décision au niveau cache. Par ailleurs, dans le cas d'une architecture constituée de processeurs hétérogènes, la modulation des fréquences des processeurs pourrait permettre de construire les arbres de décision en des temps plus ou moins équivalents de sorte à bénéficier de l'indexation des données communes entre les arbres simultanément construits.

Cas d'un système distribué. Dans le cas d'une architecture d'EI décentralisée, K-MLIO peut être distribuée en adoptant une architecture maître-esclave (exemple, l'apprentissage fédéré ou *Federated Learning* [153]) où les dispositifs esclaves appliquent le *K-means* sur les *chunks*, et le dispositif maître agrège les résultats en appliquant le *K-means* sur le *chunk* final. Deux problèmes peuvent, alors, se poser :

— La distribution des données : dans le cas où la collecte de données est distribuée sur tous les nœuds de l'architecture, les données collectées peuvent ne pas être dis-

tribuées de manière équilibrée [163] (*Not independant and identically distributed data*). Dans le cas de K-MLIO par exemple, si l'on suppose que chaque dispositif esclave applique le *K-means* partiel des données qu'il a localement collecté, cela ralentirait les calculs des *K-means* partiels tel que décrit en section 3.2.4 page 89. La solution serait alors de redistribuer les données sur les différents nœuds de l'architecture de sorte à obtenir des *chunks* équilibrés. Par ailleurs, dans le cas de RaFIO où les arbres sont amenés à être construits de manière parallèle sur plusieurs nœuds, le même problème de données redondantes, décrit pour le cas de parallélisation sur une architecture à mémoire partagée se pose. Pour répondre à ce problème, la distribution et le placement des données du *data-set* sur les nœuds selon les *bootstrap* qui leurs ont été affectés, de manière à réduire la transmission et le stockage de données redondantes peut être très intéressant. Un problème analogue portant sur le placement de données sur une architecture qui contient des producteurs et des consommateurs a été traité dans [94].

— Le *mapping* des calculs sur les différents nœuds de l'architecture : plusieurs paramètres sont à prendre en considération dans la manière de distribuer les calculs sur les dispositifs esclaves :
  — L'hétérogénéité des dispositifs de l'architecture : les dispositifs matériels qui constituent l'architecture peuvent être hétérogènes en terme de capacités de calcul, mémoire, stockage et communication [163, 4]. Cette hétérogénéité doit, donc, être prise en en compte dans le *mapping* des calculs ;
  — La consommation énergétique : le budget en énergie des dispositifs esclaves peut être différent. Par conséquent, pour maximiser le temps de disponibilité des dispositifs esclaves, la distribution des calculs devrait prendre en considération le budget en énergie restant sur les dispositifs, par rapport au coût des calculs et des communications nécessaires à leur distribution.

La problématique est alors de trouver une méthode de distribution des données et des calculs pour les applications K-MLIO/RaFIO en tenant compte de ces contraintes. Une solution possible serait, comme dans [94], de formuler ce problème sous forme d'ILP (*Integer Linear Programming*), de sorte à affecter les calculs de *K-means* partiels, en réalisant le meilleur compromis entre consommation énergétique de l'architecture et temps d'apprentissage total. Ce compromis peut, par exemple, être traduit en utilisant la métrique *Energy Delay Product* (EDP), la fonction objectif serait alors la minimisation de cette métrique, sous des contraintes de budget mémoire et énergie par esclave.

Étude d'adaptabilité des mécanismes mis en œuvre dans cette thèse à d'autres algorithmes de ML

Dans cette thèse, nous avons montré, à travers deux cas d'études, que le coût des E/S dans le cas des algorithmes de ML est important et ralentit considérablement la tâche d'apprentissage. Nous pensons que les motifs d'E/S mis en évidence dans cette thèse peuvent être similaires dans d'autres algorithmes d'apprentissage, ce qui permettrait d'utiliser les mêmes mécanismes employés dans K-MLIO et RaFIO pour réduire les E/S. Dans ce qui suit, nous énumérons les algorithmes qui nous paraissent appartenir à cette catégorie, d'après une analyse théorique du principe de fonctionnement de l'algorithme.

— Les *Gaussian Mixture Model* (GMM). En effet, cet algorithme est une généralisation de l'algorithme du *K-means*. Les observations ne sont pas assignées de manière « rigide » à des centres de *clusters* comme dans le cas du *K-means*, mais affectées à la distribution Gaussienne ayant généré chacun des *clusters* avec une certaine probabilité. L'algorithme d'apprentissage utilisé pour les GMM est le *Expectation Maximization* (EM) qui parcourt l'ensemble des observations du *data-set* à chaque itération, de la même façon que le *K-means*. Par conséquent, il serait intéressant d'appliquer le mécanisme utilisé dans K-MLIO pour réduire le volume des E/S ;

— Les *Support Vector Machine* (SVM) sont un des autres algorithmes auxquels peut s'appliquer l'approche utilisée dans K-MLIO. En effet, l'entraînement des SVM consiste à trouver les hyperplans qui séparent les observations du *data-set*. La mise en œuvre de cette méthode comprend le parcours du *data-set* sur plusieurs itérations. La cause algorithmique des E/S est, donc, similaire au cas *K-means* et pourrait être résolu par le même mécanisme que K-MLIO ;

— Le mécanisme de réorganisation des données peut être appliqué à d'autres algorithmes de ML, basés sur les arbres tels que le *Boosting*, l'*eXtreme Gradient Boosting* (XGBoost) les *Baysian Additive Regression Trees*. Plus généralement, la réorganisation des données (*data-set* ou modèle) pour augmenter la localité spatiale peut être intéressante dans le cas des algorithmes où il se dégage des sous-ensemble de données doivent être accédés en même temps.

Par ailleurs, d'autres motifs d'E/S peuvent être découverts en analysant d'autres algorithmes. À long terme, il serait très intéressant de proposer une méthode de caractérisation d'un point du vue E/S des algorithmes de ML qui permettrait d'automatiser la détection des motifs d'E/S. Pour cela, nous pouvons imaginer d'étendre un traceur d'E/S au niveau applicatif, par exemple [95], par un analyseur/classifieur de motifs. Une fois les différents motifs d'E/S détectés, il serait possible de regrouper les algorithmes de ML présentant des motifs similaires. L'intérêt d'une telle classification serait de généraliser les mécanismes à

utiliser pour chacune des classes d'algorithmes. Il serait, alors, intéressant de développer une librairie pour l'accélération des algorithmes de ML qui automatiserait l'intégration des mécanismes d'accélération selon le motif d'E/S détecté. Cette perspective a également était formulée du point de vu matériel, pour un système composable et pour le cas particulier des CNNs [43].

Pour clore cette thèse, nous concluons que l'étude que nous avons menée nous a permis de réduire considérablement le temps d'apprentissage sur nos cas d'étude, en réduisant le coût des E/S : la réduction obtenue est de 60% pour le *K-means* et de 70% pour les *Random Forests*. Dans le cadre de l'EI, où le volume de la mémoire principale est réduit, cela est très encourageant, et renforce l'idée que la révision des algorithmes de ML du point de vue transferts entre mémoire principale et stockage secondaire peut permettre une meilleure intégration de la tâche d'apprentissage au niveau des dispositifs embarqués.

Appendices

Annexe A

La carte BeagleBone Black

Dans cette annexe, nous passons en revue les caractéristiques de la carte embarquée BeagleBone Black, montrée figure A.1, et décrivons succinctement la configuration utilisée pour nos expérimentations.

Caractéristiques matérielles : La BeagleBone Black est fabriquée par Circuitco LLC à Richardson Texas et classée parmi les matériels libres conçus par BeagleBoard.org. Le tableau A.1 synthétise les caractéristiques matérielles de cette carte.

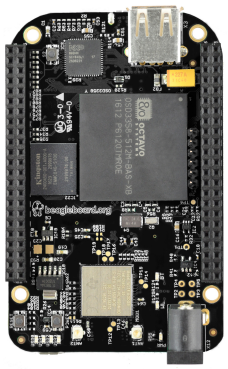


FIGURE A.1 – Carte BeagleBone Black

	Propriété
Processeur	Sitara AM3359AZCZ100 1GHz, 2000 MIPS
Mémoire SDRAM	512Mo DDR3L 606MHZ
Mémoire flash	2Go, 8bit Embedded MMC
Connecteur SD/MMC	microSD

TABLE A.1 – Propriétés de la BeagleBone Black

Configuration d'espace *swap* utilisée pour les tests : la carte BeagleBone Black a un espace RAM de 512Mo. Dans les expérimentations relatives à K-MLIO, nous avons ajouté un espace de *swap* de 2Go en créant une partition *swap* sur une carte SD selon la démarche décrite dans [20].

Annexe B

Génération de *data-sets* synthétiques

Dans ce qui suit, nous décrivons la méthode de génération de *data-sets* synthétiques utilisés pour évaluer K-MLIO et RaFIO.

B.1 Génération de *data-sets* synthétiques pour le *clustering*

Afin d'évaluer la méthode K-MLIO, nous avons eu besoin d'utiliser des *data-sets* de tailles et caractéristiques différentes. Pour ce faire, nous avons utilisé des *data-sets* synthétiques que nous avons généré en utilisant le *package ClusterGeneration* de R [111]. Plus précisément, la fonction utilisée de ce *package* est *genRandomClust* avec les paramètres suivants :

— *numClust* : nombre de *clusters* que va contenir le *data-set*. Nous avons fixé ce paramètre à 10 ;
— *numNonNoisy* : nombre de propriétés du *data-set*. Ce paramètre a été fixé à 10 ;
— *SepVal* : indice de séparation des *clusters*. Ce paramètre a été fixé à 0.01 pour le cas des données moyennement séparées (*data-set* moyennement complexe), -0.6 pour des données peu séparées (*data-set* complexe), et 0.6 pour des données séparées (*data-set* simple) ;
— *clustszind* : ce paramètre spécifie si le nombre d'observations par *cluster* à générer est le même pour tous les *clusters*. Nous avons choisi de générer des *clusters* équilibrés en terme de nombre d'observations en fixant la valeur de ce paramètre à 1. Cela permet notamment d'avoir plus de contrôle sur la taille du *data-set* ;
— *clustSizeEq* : ce paramètre spécifie le nombre d'observations par *cluster*. Nous avons généré des *data-sets* de tailles différentes en faisant varier ce paramètre ;
— *fileName* : il permet de nommer le fichier *data-set* à générer.

Les autres des paramètres n'ont pas été modifiés, et ont donc été laissés à leurs valeurs par défaut.

Pour chaque *data-set* généré, un fichier contenant les centres effectifs des *clusters* générés est créé par cette fonction. Ce fichier a été utilisé pour l'évaluation de la précision de la méthode K-MLIO, en comparant les centres obtenus avec la méthode, aux centres effectifs.

B.2 Génération de *data-sets* synthétiques pour la classification

Pour évaluer la méthode RaFIO, nous avons généré des *data-sets* synthétiques en utilisant la fonction *make_classification* du *package datasets* du *Framework Scikit-Learn*. Les paramètres utilisés pour cette génération sont les suivants :

— *n_samples* : il permet de fixer le nombre d'observations du *data-set*. Nous avons fait varier ce paramètre de sorte à effectuer des expérimentations sous différentes contraintes mémoire ;

— *n_features* : ce paramètre sert à fixer le nombre de propriétés (dimension des observations). Il a été utilisé pour générer des *data-sets* de dimension différentes ;

— *n_classes* : il sert à fixer le nombre de classes possibles. Pour les *data-sets* synthétiques, nous avons fixé le nombre de classes à 2.

Les autres des paramètres n'ont pas été modifiés, et ont donc été laissés à leurs valeurs par défaut.

Références bibliographiques

[1] Mohiuddin AHMED, Raihan SERAJ et Syed Mohammed Shamsul ISLAM. "The k-means Algorithm : A Comprehensive Survey and Performance Evaluation". In : *Electronics* 9.8 (2020). ISSN : 2079-9292. DOI : 10.3390/electronics9081295. URL : https://www.mdpi.com/2079-9292/9/8/1295.

[2] Marc-Ismaël AKODJÈNOU-JEANNIN, Kavé SALAMATIAN et Patrick GALLINARI. "Flexible Grid-Based Clustering". In : *Knowledge Discovery in Databases : PKDD 2007*. Sous la dir. de Joost N. KOK, Jacek KORONACKI, Ramon Lopez de MANTARAS, Stan MATWIN, Dunja MLADENIČ et Andrzej SKOWRON. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, p. 350-357. ISBN : 978-3-540-74976-9.

[3] Antonio ALARCÓN-PAREDES, Victor FRANCISCO-GARCÍA, Iris P. GUZMÁN-GUZMÁN, Jessica CANTILLO-NEGRETE, René E. CUEVAS-VALENCIA et Gustavo A. ALONSO-SILVERIO. "An IoT-Based Non-Invasive Glucose Level Monitoring System Using Raspberry Pi". In : *Applied Sciences* 9.15 (2019). ISSN : 2076-3417. DOI : 10.3390/app9153046. URL : https://www.mdpi.com/2076-3417/9/15/3046.

[4] Mohammed ALEDHARI, Rehma RAZZAK, Reza M PARIZI et Fahad SAEED. "Federated learning : A survey on enabling technologies, protocols, and applications". In : *IEEE Access* 8 (2020), p. 140699-140725.

[5] Andreea ANGHEL, Nikolas IOANNOU, Thomas PARNELL, Nikolaos PAPANDREOU, Celestine MENDLER-DÜNNER et Haris POZIDIS. "Breadth-first, Depth-next Training of Random Forests". In : *ArXiv* abs/1910.06853 (2019).

[6] Shaahin ANGIZI, Zhezhi HE, Farhana PARVEEN et Deliang FAN. "IMCE : Energy-efficient bit-wise in-memory convolution engine for deep neural network". In : *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2018, p. 111-116. DOI : 10.1109/ASPDAC.2018.8297291.

[7] Chloé-Agathe AZENCOTT. *Introduction au Machine Learning*. Français. Dunod. InfoSup. Juil. 2019. ISBN : 2100780808. URL : https://www.dunod.com/sciences-techniques/introduction-au-machine-learning-0 (visité le 29/06/2021).

[8] Daniel BARBARÁ et Ping CHEN. "Using the fractal dimension to cluster datasets". In : *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2000, p. 260-264.

[9] Sathwika BAVIKADI, Purab Ranjan SUTRADHAR, Khaled N. KHASAWNEH, Amlan GANGULY et Sai Manoj PUDUKOTAI DINAKARRAO. "A Review of In-Memory Computing Architectures for Machine Learning Applications". In : *Proceedings of the 2020 on Great Lakes Symposium on VLSI*. GLSVLSI '20. Virtual Event, China : Association for Computing Machinery, 2020, 89–94. ISBN : 9781450379441. DOI : 10.1145/3386263.3407649. URL : https://doi.org/10.1145/3386263.3407649.

[10] Hadjer BENMEZIANE, Kaoutar El MAGHRAOUI, Hamza OUARNOUGHI, Smail NIAR, Martin WISTUBA et Naigang WANG. "A comprehensive survey on hardware-aware neural architecture search". In : *arXiv preprint arXiv :2101.09336* (2021).

[11] Yahia BENMOUSSA, Jalil BOUKHOBZA, Eric SENN et Djamel BENAZZOUZ. "Energy consumption modeling of H. 264/AVC video decoding for GPP and DSP". In : *2013 Euromicro Conference on Digital System Design*. IEEE. 2013, p. 890-897.

[12] Mohammed BEY AHMED KHERNACHE, Yahia BENMOUSSA, Jalil BOUKHOBZA et Daniel MENARD. "HEVC hardware vs software decoding : An objective energy consumption analysis and comparison". In : *Journal of Systems Architecture* 115 (2021), p. 102004. ISSN : 1383-7621. DOI : https://doi.org/10.1016/j.sysarc.2021.102004. URL : https://www.sciencedirect.com/science/article/pii/S1383762121000199.

[13] Thomas BOTTESCH, Thomas BÜHLER et Markus KÄCHELE. "Speeding up k-means by approximating Euclidean distances via block vectors". In : *International Conference on Machine Learning*. PMLR. 2016, p. 2578-2586.

[14] Leon BOTTOU et Yoshua BENGIO. "Convergence properties of the K-Means algorithms". In : *Advances in Neural Information Processing Systems*. 1995, p. 585-592.

[15] Jalil BOUKHOBZA et Pierre OLIVIER. *Flash Memory Integration : Performance and Energy Issues, 1st Edition*. ISTE Press - Elsevier, 2017. ISBN : 978-1-78548-124-6. URL : https://www.sciencedirect.com/science/book/9781785481246.

[16] Jalil BOUKHOBZA, Stéphane RUBINI, Renhai CHEN et Zili SHAO. "Emerging NVM : A Survey on Architectural Integration and Research Challenges". In : *ACM Trans. Des. Autom. Electron. Syst.* 23.2 (nov. 2017). ISSN : 1084-4309. DOI : 10.1145/3131848. URL : https://doi.org/10.1145/3131848.

[17] Daniel BOVET et Marco CESATI. *Understanding The Linux Kernel*. Oreilly amp ; Associates Inc, 2005. ISBN : 0596005652.

[18] Sérgio BRANCO, André G. FERREIRA et Jorge CABRAL. "Machine Learning in Resource-Scarce Embedded Systems, FPGAs, and End-Devices : A Survey". In : *Electronics* 8.11 (2019). ISSN : 2079-9292. DOI : 10.3390/electronics8111289. URL : https://www.mdpi.com/2079-9292/8/11/1289.

[19] Leo BREIMAN. "Bagging predictors". In : *Machine learning* 24.2 (1996), p. 123-140.

[20] Paul BUPE. *Adding swap memory to the Beaglebone Black*. 2021. URL : https://paulbupejr.com/adding-swap-memory-to-the-beaglebone-black/.

[21] *CAFFE2*. URL : https://caffe2.ai/.

[22] Sen CAO, Yazhou LIU, Pongsak LASANG et Shengmei SHEN. "Detecting the objects on the road using modular lightweight network". In : *arXiv preprint arXiv :1811.06641* (2018).

[23] Robert CATTRAL et Franz OPPACHER. "Discovering rules in the poker hand dataset". In : *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. 2007, p. 1870-1870.

[24] M. Emre CELEBI, Hassan A. KINGRAVI et Patricio A. VELA. "A comparative study of efficient initialization methods for the k-means clustering algorithm". In : *Expert Systems with Applications* 40.1 (2013), p. 200-210. ISSN : 0957-4174. DOI : https://doi.org/10.1016/j.eswa.2012.07.021. URL : https://www.sciencedirect.com/science/article/pii/S0957417412008767.

[25] José E CHACÓN et Ana I RASTROJO. "Minimum adjusted Rand index for two clusterings of a given size". In : *Advances in Data Analysis and Classification* (2022), p. 1-9.

[26] Olivier CHAPELLE, Bernhard SCHÖLKOPF et Alexander ZIEN. *Semi-supervised Learning. Adaptive computation and machine learning series*. 2006.

[27] Tianqi CHEN, Mu LI, Yutian LI, Min LIN, Naiyan WANG, Minjie WANG, Tianjun XIAO, Bing XU, Chiyuan ZHANG et Zheng ZHANG. "Mxnet : A flexible and efficient machine learning library for heterogeneous distributed systems". In : *arXiv preprint arXiv :1512.01274* (2015).

[28] Yu-Hsin CHEN, Tushar KRISHNA, Joel S. EMER et Vivienne SZE. "Eyeriss : An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks". In : *IEEE Journal of Solid-State Circuits* 52.1 (2017), p. 127-138. DOI : 10.1109/JSSC.2016.2616357.

[29] Yu-Hsin CHEN, Tien-Ju YANG, Joel EMER et Vivienne SZE. "Eyeriss v2 : A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices". In : *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.2 (2019), p. 292-308. DOI : 10.1109/JETCAS.2019.2910232.

[30] Yunji CHEN, Tianshi CHEN, Zhiwei XU, Ninghui SUN et Olivier TEMAM. "DianNao Family : Energy-Efficient Hardware Accelerators for Machine Learning". In : *Commun. ACM* 59.11 (2016), 105–112. ISSN : 0001-0782. DOI : 10.1145/2996864. URL : https://doi.org/10.1145/2996864.

[31] Daniel CLOAREC et Anne COUËTIL. *Vous avez dit marétique ? Des opportunités à saisir à la confluence de la mer et du numérique en Bretagne*. Conseil économique, social et environnemental régional de Bretagne, 2019. URL : https://ceser.bretagne.bzh/upload/docs/application/pdf/2019-10/ceser_vous_avez_dit_maretique_rapport_web.pdf.

[32] Andreea Ancuta CORICI. *Edge intelligence*. IEC, 2017. URL : https://storage-iecwebsite-prd-iec-ch.s3.eu-west-1.amazonaws.com/2019-09/content/media/files/iec_wp_edge_intelligence_en_lr.pdf.

[33] Kevin CROWSTON. "Amazon Mechanical Turk : A Research Tool for Organizations and Information Systems Scholars". In : *Shaping the Future of ICT Research. Methods and Approaches*. Sous la dir. d'Anol BHATTACHERJEE et Brian FITZGERALD. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, p. 210-221. ISBN : 978-3-642-35142-6.

[34] Huimin CUI, Gong RUAN, Jingling XUE, Rui XIE, Lei WANG et Xiaobing FENG. "A collaborative divide-and-conquer K-means clustering algorithm for processing large data". In : *Proceedings of the 11th ACM Conference on Computing Frontiers* (2014).

[35] Salvatore CUOMO, V DE ANGELIS, Gennaro FARINA, Livia MARCELLINO et Gerardo TORALDO. "A GPU-accelerated parallel K-means algorithm". In : *Computers Electrical Engineering* 75 (2019), p. 262-274. ISSN : 0045-7906. DOI : https://doi.org/10.1016/j.compeleceng.2017.12.002. URL : https://www.sciencedirect.com/science/article/pii/S0045790617327994.

[36] Jean-Philippe DIGUET. *Programme SAMM "Systèmes Autonomes en Milieu Mari-me" - Appel à manifes-ons d'intérêt à des-na-on des PME Bretonnes*. URL : http://labsticc.extrazimut.net/modules/kameleon/upload/samm-flyer.pdf.

[37] Yufei DING, Yue ZHAO, Xipeng SHEN, Madanlal MUSUVATHI et Todd MYTKOWICZ. "Yinyang K-Means : A Drop-In Replacement of the Classic K-Means with Consistent Speedup". In : *Proceedings of the 32nd International Conference on Machine Learning*. Sous la dir. de Francis BACH et David BLEI. T. 37. Proceedings of Machine Learning Research. Lille, France : PMLR, 2015, p. 579-587. URL : https://proceedings.mlr.press/v37/ding15.html.

[38] Jaeyoung DO, Yang-Suk KEE, Jignesh M. PATEL, Chanik PARK, Kwanghyun PARK et David J. DEWITT. "Query Processing on Smart SSDs : Opportunities and Challenges". In : *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD '13. New York, New York, USA : Association for Computing Machinery, 2013, 1221–1230. ISBN : 9781450320375. DOI : 10.1145/2463676.2465295. URL : https://doi.org/10.1145/2463676.2465295.

[39] Jonathan DRAKE et Greg HAMERLY. "Accelerated k-means with adaptive distance bounds". In : *5th NIPS workshop on optimization for machine learning*. T. 8. 2012.

[40] Zidong DU, Robert FASTHUBER, Tianshi CHEN, Paolo IENNE, Ling LI, Tao LUO, Xiaobing FENG, Yunji CHEN et Olivier TEMAM. "ShiDianNao : Shifting vision processing closer to the sensor". In : *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 2015, p. 92-104. DOI : 10.1145/2749469.2750389.

[41] Dheeru DUA et Casey GRAFF. *UCI Machine Learning Repository*. 2017.

[42] Elsa DUPRAZ, Dominique PASTOR et François-Xavier SOCHELEAU. "Decentralized Clustering on Compressed Data without Prior Knowledge of the Number of Clusters". In : *arXiv preprint arXiv :1807.04566* (2018).

[43] Kaoutar EL MAGHRAOUI, Lorraine M HERGER, Chekuri CHOUDARY, Kim TRAN, Todd DESHANE et David HANSON. "Performance Analysis of Deep Learning Workloads on a Composable System". In : *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2021, p. 951-954.

[44] Charles ELKAN. "Using the triangle inequality to accelerate k-means". In : *Proceedings of the 20th international conference on Machine Learning (ICML-03)*. 2003, p. 147-153.

[45] Martin ESTER, Hans-Peter KRIEGEL, Jörg SANDER et Xiaowei XU. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In : *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon : AAAI Press, 1996, 226–231.

[46] Syed FAIAZUDDIN, M.V. LAKSHMAIAH, K. Tanveer ALAM et M. RAVIKIRAN. "IoT based Indoor Air Quality Monitoring system using Raspberry Pi4". In : *2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. 2020, p. 714-719. DOI : 10.1109/ICECA49313.2020.9297442.

[47] Emanuel FALKENAUER. "On Method Overfitting". In : *Journal of Heuristics* 4 (1998), p. 281-287.

[48] Biyi FANG, Xiao ZENG et Mi ZHANG. "NestDNN : Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision". In : *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. MobiCom '18. New Delhi, India : Association for Computing Machinery, 2018, 115–127. ISBN : 9781450359030. DOI : 10.1145/3241539.3241559. URL : https://doi.org/10.1145/3241539.3241559.

[49] Marin FERECATU. *Cours - arbres de décision*. URL : http://cedric.cnam.fr/vertigo/cours/ml2/coursArbresDecision.html#elagage-et-gestion-de-donnees-manquantes.

[50] Douglas H FISHER. "Knowledge acquisition via incremental conceptual clustering". In : *Machine learning* 2.2 (1987), p. 139-172.

[51] Abdoulaye GAMATIÉ, Xin AN, Ying ZHANG, An KANG et Gilles SASSATELLI. "Empirical model-based performance prediction for application mapping on multicore architectures". In : *Journal of Systems Architecture* 98 (2019), p. 1-16.

[52] Abdoulaye GAMATIÉ, Guillaume DEVIC, Gilles SASSATELLI, Stefano BERNABOVI, Philippe NAUDIN et Michael CHAPMAN. "Towards energy-efficient heterogeneous multicore architectures for edge computing". In : *IEEE Access* 7 (2019), p. 49474-49491.

[53] Victor GONZALEZ-HUITRON, José A. LEÓN-BORGES, A.E. RODRIGUEZ-MATA, Leonel Ernesto AMABILIS-SOSA, Blenda RAMÍREZ-PEREDA et Hector RODRIGUEZ. "Disease detection in tomato leaves via CNN with lightweight architectures implemented in Raspberry Pi 4". In : *Computers and Electronics in Agriculture* 181 (2021), p. 105951. ISSN : 0168-1699. DOI : https://doi.org/10.1016/j.compag.

2020.105951. URL : https://www.sciencedirect.com/science/article/pii/S0168169920331562.

[54] Frederik GOSSEN et Bernhard STEFFEN. "Large Random Forests : Optimisation for Rapid Evaluation". In : *CoRR* abs/1912.10934 (2019). arXiv : 1912.10934. URL : http://arxiv.org/abs/1912.10934.

[55] Paulene GOVENDER et Venkataraman SIVAKUMAR. "Application of k-means and hierarchical clustering techniques for analysis of air pollution : A review (1980–2019)". In : *Atmospheric Pollution Research* 11.1 (2020), p. 40-56. ISSN : 1309-1042. DOI : https://doi.org/10.1016/j.apr.2019.09.009. URL : https://www.sciencedirect.com/science/article/pii/S1309104219304556.

[56] Boncheol GU, Andre S. YOON, Duck-Ho BAE, Insoon JO, Jinyoung LEE, Jonghyun YOON, Jeong-Uk KANG, Moonsang KWON, Chanho YOON, Sangyeun CHO, Jaeheon JEONG et Duckhyun CHANG. "Biscuit : A Framework for Near-Data Processing of Big Data Workloads". In : *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 2016, p. 153-165. DOI : 10.1109/ISCA.2016.23.

[57] Sudipto GUHA, Adam MEYERSON, Nina MISHRA, Rajeev MOTWANI et Liadan O'CALLAGHAN. "Clustering Data Streams : Theory and Practice". In : *IEEE Trans. on Knowl. and Data Eng.* 15.3 (mar. 2003), 515–528. ISSN : 1041-4347. DOI : 10.1109/TKDE.2003.1198387. URL : https://doi.org/10.1109/TKDE.2003.1198387.

[58] Sudipto GUHA, Rajeev RASTOGI et Kyuseok SHIM. "CURE : An Efficient Clustering Algorithm for Large Databases". In : *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*. SIGMOD '98. Seattle, Washington, USA : Association for Computing Machinery, 1998, 73–84. ISBN : 0897919955. DOI : 10.1145/276304.276312. URL : https://doi.org/10.1145/276304.276312.

[59] Chirag GUPTA, Arun Sai SUGGALA, Ankit GOYAL, Harsha Vardhan SIMHADRI, Bhargavi PARANJAPE, Ashish KUMAR, Saurabh GOYAL, Raghavendra UDUPA, Manik VARMA et Prateek JAIN. "ProtoNN : Compressed and Accurate KNN for Resource-Scarce Devices". In : *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia : JMLR.org, 2017, 1331–1340.

[60] Saransh GUPTA, Behnam KHALEGHI, Sahand SALAMAT, Justin MORRIS, Ranganathan RAMKUMAR, Jeffrey YU, Aniket TIWARI, Jaeyoung KANG, Mohsen IMANI, Baris AKSANLI et Tajana Šimunić ROSING. "Store-n-Learn : Classification and

Clustering with Hyperdimensional Computing across Flash Hierarchy". In : *ACM Trans. Embed. Comput. Syst.* (2021). Just Accepted. ISSN : 1539-9087. DOI : 10.1145/3503541. URL : https://doi.org/10.1145/3503541.

[61] Christophe GUYEUX, Stéphane CHRÉTIEN, Gaby BOU TAYEH, Jacques DEMERJIAN et Jacques BAHI. "Introducing and Comparing Recent Clustering Methods for Massive Data Management in the Internet of Things". In : *Journal of Sensor and Actuator Networks* 8.4 (2019). ISSN : 2224-2708. DOI : 10.3390/jsan8040056. URL : https://www.mdpi.com/2224-2708/8/4/56.

[62] Greg HAMERLY. "Making k-means even faster". In : *Proceedings of the 2010 SIAM international conference on data mining*. SIAM. 2010, p. 130-140.

[63] Seungyeop HAN, Haichen SHEN, Matthai PHILIPOSE, Sharad AGARWAL, Alec WOLMAN et Arvind KRISHNAMURTHY. "MCDNN : An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints". In : *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '16. Singapore, Singapore : Association for Computing Machinery, 2016, 123–136. ISBN : 9781450342698. DOI : 10.1145/2906388.2906396. URL : https://doi.org/10.1145/2906388.2906396.

[64] Yu Ting HO, Chun-Feng WU, Ming-Chang YANG, Tseng-Yi CHEN et Yuan-Hao CHANG. "Replanting Your Forest : NVM-friendly Bagging Strategy for Random Forest". In : *2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. 2019, p. 1-6. DOI : 10.1109/NVMSA.2019.8863525.

[65] Yun Chao HU, Milan PATEL, Dario SABELLA, Nurit SPRECHER et Valerie YOUNG. "Mobile edge computing—A key technology towards 5G". In : *ETSI white paper* 11.11 (2015), p. 1-16.

[66] Bruce JACOB, Spencer NG et David WANG. *Memory Systems : Cache, DRAM, Disk*. Jan. 2008. ISBN : 978-0-12-379751-3.

[67] Anil K JAIN. "Data clustering : 50 years beyond K-means". In : *Pattern recognition letters* 31.8 (2010), p. 651-666.

[68] Liangxiao JIANG, Zhihua CAI, Dianhong WANG et Siwei JIANG. "Survey of Improving K-Nearest-Neighbor for Classification". In : *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*. T. 1. 2007, p. 679-683. DOI : 10.1109/FSKD.2007.552.

[69] Insoon JO, Duck-Ho BAE, Andre S. YOON, Jeong-Uk KANG, Sangyeun CHO, Daniel D. G. LEE et Jaeheon JEONG. "YourSQL : A High-Performance Database System Leveraging in-Storage Computing". In : *Proc. VLDB Endow.* 9.12 (2016), 924–935. ISSN : 2150-8097. DOI : 10.14778/2994509.2994512. URL : https://doi.org/10.14778/2994509.2994512.

[70] Duseok KANG, Jinwoo OH, Jongwoo CHOI, Youngmin YI et Soonhoi HA. "Scheduling of Deep Learning Applications Onto Heterogeneous Processors in an Embedded Device". In : *IEEE Access* 8 (2020), p. 43980-43991. DOI : 10.1109/ACCESS.2020.2977496.

[71] George KARYPIS, Eui-Hong HAN et Vipin KUMAR. "Chameleon : hierarchical clustering using dynamic modeling". In : *Computer* 32.8 (1999), p. 68-75. DOI : 10.1109/2.781637.

[72] Leonard KAUFMAN et Peter J ROUSSEEUW. "Clustering large applications (Program CLARA)". In : *Finding groups in data : an introduction to cluster analysis* (2008), p. 126-146.

[73] Pavleen KAUR, Ravinder KUMAR et Munish KUMAR. "A healthcare monitoring system using random forest and internet of things (IoT)". In : *Multimedia Tools and Applications* 78.14 (2019), p. 19905-19916.

[74] Trupti M KODINARIYA et Prashant R MAKWANA. "Review on determining number of Cluster in K-Means Clustering". In : *International Journal* 1.6 (2013), p. 90-95.

[75] Gunjae KOO, Kiran Kumar MATAM, Te I., H.V. Krishna Giri NARRA, Jing LI, Hung-Wei TSENG, Steven SWANSON et Murali ANNAVARAM. "Summarizer : Trading Communication with Computing Near Storage". In : *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2017, p. 219-231.

[76] Ashish KUMAR, Saurabh GOYAL et Manik VARMA. "Resource-Efficient Machine Learning in 2 KB RAM for the Internet of Things". In : *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia : JMLR.org, 2017, 1935–1944.

[77] Mohit KUMAR, Xingzhou ZHANG, Liangkai LIU, Yifan WANG et Weisong SHI. "Energy-Efficient Machine Learning on the Edges". In : *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2020, p. 912-921. DOI : 10.1109/IPDPSW50202.2020.00153.

[78] Charles LEECH, Yordan P. RAYKOV, Emre OZER et Geoff V. MERRETT. "Real-time room occupancy estimation with Bayesian machine learning using a single PIR sensor and microcontroller". In : *2017 IEEE Sensors Applications Symposium (SAS)*. 2017, p. 1-6. DOI : 10.1109/SAS.2017.7894091.

[79] Shuangchen LI, Alvin Oliver GLOVA, Xing HU, Peng GU, Dimin NIU, Krishna T. MALLADI, Hongzhong ZHENG, Bob BRENNAN et Yuan XIE. "SCOPE : A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator". In : *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2018, p. 696-709. DOI : 10.1109/MICRO.2018.00062.

[80] Shuangchen LI, Cong XU, Qiaosha ZOU, Jishen ZHAO, Yu LU et Yuan XIE. "Pinatubo : A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories". In : *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2016, p. 1-6. DOI : 10.1145/2897937.2898064.

[81] You LI, Kaiyong ZHAO, Xiaowen CHU et Jiming LIU. "Speeding up K-Means Algorithm by GPUs". In : *2010 10th IEEE International Conference on Computer and Information Technology*. 2010, p. 115-122. DOI : 10.1109/CIT.2010.60.

[82] Yue LIU et Zhiqiang GE. "Weighted random forests for fault classification in industrial processes with hierarchical clustering model selection". In : *Journal of Process Control* 64 (2018), p. 62-70. ISSN : 0959-1524. DOI : https://doi.org/10.1016/j.jprocont.2018.02.005. URL : https://www.sciencedirect.com/science/article/pii/S0959152418300271.

[83] Stuart P. LLOYD. "Least squares quantization in PCM". In : *IEEE Trans. Information Theory* 28 (1982), p. 129-136.

[84] Moe LONG et FOLLOW. *Best Single-Board Computers for Every Use and Budget in 2021*. 2021. URL : https://www.electromaker.io/blog/article/best-single-board-computers (visité le 24/06/2021).

[85] Yufei MA, Yu CAO, Sarma VRUDHULA et Jae-sun SEO. "Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks". In : *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '17. Monterey, California, USA : Association for Computing Machinery, 2017, 45–54. ISBN : 9781450343541. DOI : 10.1145/3020078.3021736. URL : https://doi.org/10.1145/3020078.3021736.

[86] James MACQUEEN et al. "Some methods for classification and analysis of multivariate observations". In : *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. T. 1. 14. Oakland, CA, USA. 1967, p. 281-297.

[87] Vikram Sharma MAILTHODY, Zaid QURESHI, Weixin LIANG, Ziyan FENG, Simon Garcia de GONZALO, Youjie LI, Hubertus FRANKE, Jinjun XIONG, Jian HUANG et Wen-mei HWU. "DeepStore : In-Storage Acceleration for Intelligent Queries". In : *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO '52. Columbus, OH, USA : Association for Computing Machinery, 2019, 224–238. ISBN : 9781450369381. DOI : 10.1145/3352460.3358320. URL : https://doi.org/10.1145/3352460.3358320.

[88] Paul MENAGE. *CGROUPS*. https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt, last accessed on 28/07/20. 2004.

[89] P. MIRON, M. J. OLASCOAGA, F. J. BERON-VERA, N. F. PUTMAN, J. TRIÑANES, R. LUMPKIN et G. J. GONI. "Clustering of Marine-Debris- and Sargassum-Like Drifters Explained by Inertial Particle Dynamics". In : *Geophysical Research Letters* 47.19 (2020). e2020GL089874 10.1029/2020GL089874, e2020GL089874. DOI : https://doi.org/10.1029/2020GL089874. eprint : https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2020GL089874. URL : https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2020GL089874.

[90] Tom M MITCHELL et al. "Machine learning. 1997". In : *Burr Ridge, IL : McGraw Hill* 45.37 (1997), p. 870-877.

[91] Onur MUTLU. "Intelligent Architectures for Intelligent Computing Systems". In : *CoRR* abs/2012.12381 (2020). arXiv : 2012.12381. URL : https://arxiv.org/abs/2012.12381.

[92] Onur MUTLU, Saugata GHOSE, Juan GÓMEZ-LUNA et Rachata AUSAVARUNGNIRUN. *Processing Data Where It Makes Sense : Enabling In-Memory Computation*. 2019. arXiv : 1903.03988 [cs.AR].

[93] Mohammed Islam NAAS, Philippe Raipin PARVEDY, Jalil BOUKHOBZA et Laurent LEMARCHAND. "iFogStor : An IoT Data Placement Strategy for Fog Infrastructure". In : *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. 2017, p. 97-104. DOI : 10.1109/ICFEC.2017.15.

[94] Mohammed Islam NAAS, Philippe Raipin PARVEDY, Jalil BOUKHOBZA et Laurent LEMARCHAND. "iFogStor : an IoT data placement strategy for fog infrastructure".

In : *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. IEEE. 2017, p. 97-104.

[95] Mohammed Islam NAAS, François TRAHAY, Alexis COLIN, Pierre OLIVIER, Stéphane RUBINI, Frank SINGHOFF et Jalil BOUKHOBZA. "EZIOTracer : unifying kernel and user space I/O tracing for data-intensive applications". In : *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems*. 2021, p. 1-11.

[96] Gagandeep Singh NAGPAL, Gagandeep SINGH, Jappreet SINGH et Nishant YADAV. "Facial detection and recognition using OPENCV on Raspberry Pi Zero". In : *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. IEEE. 2018, p. 945-950.

[97] Disruptor By Patrick NELSON et Patrick NELSON. *One autonomous car will use 4,000 GB of data per day*. 2016. URL : https://www.networkworld.com/article/3147892/one-autonomous-car-will-use-4000-gb-of-dataday.html.

[98] Raymond NG et Jiawei HAN. "CLARANS : A method for clustering objects for spatial data mining". In : *Knowledge and Data Engineering, IEEE Transactions on* 14 (oct. 2002), p. 1003-1016. DOI : 10.1109/TKDE.2002.1033770.

[99] Hai-Long NGUYEN, Yew-Kwong WOON et Wee Keong NG. "A Survey on Data Stream Clustering and Classification". In : *Knowledge and Information Systems* 45 (déc. 2014). DOI : 10.1007/s10115-014-0808-1.

[100] Pierre OLIVIER, Jalil BOUKHOBZA, Eric SENN et Hamza OUARNOUGHI. "A Methodology for Estimating Performance and Power Consumption of Embedded Flash File Systems". In : *ACM Trans. Embed. Comput. Syst.* 15.4 (2016). ISSN : 1539-9087. DOI : 10.1145/2903139. URL : https://doi.org/10.1145/2903139.

[101] Carlos ORDONEZ et Edward OMIECINSKI. "Efficient disk-based K-means clustering for relational databases". In : *IEEE Transactions on Knowledge and Data Engineering* 16 (2004), p. 909-921.

[102] Kasey PANETTA. *5 trends emerge in Gartner Hype Cycle for Emerging Technologies 2018*. 2019. URL : https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018 (visité le 08/02/2022).

[103] Hae-Sang PARK et Chi-Hyuck JUN. "A simple and fast algorithm for K-medoids clustering". In : *Expert systems with applications* 36.2 (2009), p. 3336-3341.

[104] Greig PAUL et James IRVINE. "Privacy Implications of Wearable Health Devices". In : *Proceedings of the 7th International Conference on Security of Information and Networks*. SIN '14. Glasgow, Scotland, UK : Association for Computing Machinery, 2014, 117–121. ISBN : 9781450330336. DOI : 10.1145/2659651.2659683. URL : https://doi.org/10.1145/2659651.2659683.

[105] Fabian PEDREGOSA, Gaël VAROQUAUX, Alexandre GRAMFORT, Vincent MICHEL, Bertrand THIRION, Olivier GRISEL, Mathieu BLONDEL, Peter PRETTENHOFER, Ron WEISS, Vincent DUBOURG et al. "Scikit-learn : Machine Learning in Python". In : *Journal of Machine Learning Research* 12 (2011), p. 2825-2830.

[106] Ella PELTONEN, Mehdi BENNIS, Michele CAPOBIANCO, Merouane DEBBAH, Aaron DING, Felipe GIL-CASTIÑEIRA, Marko JURMU, Teemu KARVONEN, Markus KELANTI, Adrian KLIKS, Teemu LEPPÄNEN, Lauri LOVÉN, Tommi MIKKONEN, Ashwin RAO, Sumudu SAMARAKOON, Kari SEPPÄNEN, Paweł SROKA, Sasu TARKOMA et Tingting YANG. *6G White Paper on Edge Intelligence*. 2020. arXiv : 2004.14850 [cs.DC].

[107] Pierre-Yves PÉNEAU. "Intégration de technologies de mémoires non volatiles émergentes dans la hiérarchie de caches pour améliorer l'efficacité énergétique". Theses. Université Montpellier, oct. 2018. URL : https://tel.archives-ouvertes.fr/tel-01957250.

[108] *Pytorch*. URL : https://pytorch.org/.

[109] Jiantao QIU, Jie WANG, Song YAO, Kaiyuan GUO, Boxun LI, Erjin ZHOU, Jincheng YU, Tianqi TANG, Ningyi XU, Sen SONG, Yu WANG et Huazhong YANG. "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network". In : *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '16. Monterey, California, USA : Association for Computing Machinery, 2016, 26–35. ISBN : 9781450338561. DOI : 10.1145/2847263.2847265. URL : https://doi.org/10.1145/2847263.2847265.

[110] Junfei QIU, Qihui WU, Guoru DING, Yuhua XU et Shuo FENG. "A survey of machine learning for big data processing". In : *EURASIP Journal on Advances in Signal Processing* 2016 (2016), p. 1-16.

[111] Weiliang QIU, Harry JOE et Maintainer Weiliang QIU. *Package 'clusterGeneration'*. 2015.

[112] R CORE TEAM. *R : A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2018. URL : https://www.R-project.org/.

[113] Sudipto Guha RAJEEV, Rajeev RASTOGI et Kyuseok SHIM. "ROCK : a robust clustering algorithm for categorical attributes". In : *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*. 1999, p. 512-521. DOI : 10.1109/ICDE.1999.754967.

[114] Kishansingh RAJPUT, Bhavesh OZA, STUDENT et PROFESSOR. "A Comparative Study of Classification Techniques in Data Mining". In : t. 5. Sept. 2017, p. 154-163.

[115] Carl Edward RASMUSSEN. "The Infinite Gaussian Mixture Model". In : *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS'99. Denver, CO : MIT Press, 1999, 554–560.

[116] David REINSEL, John RYDNING et John F. GANTZ. *Worldwide Global DataSphere Forecast, 2020–2024 : The COVID-19 Data Bump and the Future of Data Growth*. 2020. URL : https://www.idc.com/getdoc.jsp?containerId=US44797920.

[117] Paulo Angelo Alves RESENDE et André Costa DRUMMOND. "A Survey of Random Forest Based Methods for Intrusion Detection Systems". In : *ACM Comput. Surv.* 51.3 (2018). ISSN : 0360-0300. DOI : 10.1145/3178582. URL : https://doi.org/10.1145/3178582.

[118] Mauro RIBEIRO, Katarina GROLINGER et Miriam A.M. CAPRETZ. "MLaaS : Machine Learning as a Service". In : *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. 2015, p. 896-902. DOI : 10.1109/ICMLA.2015.152.

[119] Mayra RODRIGUEZ, Cesar COMIN, Dalcimar CASANOVA, Odemir BRUNO, Diego AMANCIO, Francisco RODRIGUES et Luciano da F. COSTA. "Clustering Algorithms : A Comparative Approach". In : *PLOS ONE* 14 (déc. 2016). DOI : 10.1371/journal.pone.0210236.

[120] Julius ROEDER, Benjamin ROUXEL, Sebastian ALTMEYER et Clemens GRELCK. "Energy-Aware Scheduling of Multi-Version Tasks on Heterogeneous Real-Time Systems". In : *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. SAC '21. Virtual Event, Republic of Korea : Association for Computing Machinery, 2021, 501–510. ISBN : 9781450381048. DOI : 10.1145/3412841.3441930. URL : https://doi.org/10.1145/3412841.3441930.

[121] Lior ROKACH et Oded MAIMON. "Top-down induction of decision trees classifiers - a survey". In : *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 35.4 (2005), p. 476-487. DOI : 10.1109/TSMCC.2004.843247.

[122] David S ROSENBERG. "Bagging and Random Forests". In : (2017).

[123] Paul ROSERO, Vivian BATISTA, Edwin ROSERO, Edgar JARAMILLO, Jorge CARAGUAY-PROCEL, José PIJAL-ROJAS et Diego PELUFFO. "Intelligence in Embedded Systems : Overview and Applications : Volume 1". In : jan. 2019, p. 874-883. ISBN : 978-3-030-02685-1. DOI : 10.1007/978-3-030-02686-8_65.

[124] Zhenyuan RUAN, Tong HE et Jason CONG. "INSIDER : Designing In-Storage Computing System for Emerging High-Performance Drive". In : *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA : USENIX Association, juil. 2019, p. 379-394. ISBN : 978-1-939133-03-8. URL : https://www.usenix.org/conference/atc19/presentation/ruan.

[125] Jorge M SANTOS et Mark EMBRECHTS. "On the use of the adjusted rand index as a metric for evaluating supervised classification". In : *International conference on artificial neural networks*. Springer. 2009, p. 175-184.

[126] Paul-Edouard SARLIN, Frédéric DEBRAINE, Marcin DYMCZYK, Roland SIEGWART et Cesar CADENA. "Leveraging deep visual descriptors for hierarchical efficient localization". In : *Conference on Robot Learning*. PMLR. 2018, p. 456-465.

[127] Robert E SCHAPIRE. "The strength of weak learnability". In : *Machine learning* 5.2 (1990), p. 197-227.

[128] David SCULLEY. "Web-scale k-means clustering". In : *Proceedings of the 19th international conference on World wide web*. 2010, p. 1177-1178.

[129] Roded SHARAN et Ron SHAMIR. "CLICK : A Clustering Algorithm with Applications to Gene Expression Analysis". In : *Proceedings / ... International Conference on Intelligent Systems for Molecular Biology ; ISMB. International Conference on Intelligent Systems for Molecular Biology* 8 (fév. 2000), p. 307-16.

[130] Liang SHI, Chun Jason XUE, Jingtong HU, Wei-Che TSENG, Xuehai ZHOU et Edwin H.-M. SHA. "Write Activity Reduction on Flash Main Memory via Smart Victim Cache". In : *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI*. GLSVLSI '10. Providence, Rhode Island, USA : Association for Computing Machinery, 2010, 91–94. ISBN : 9781450300124. DOI : 10.1145/1785481.1785503. URL : https://doi.org/10.1145/1785481.1785503.

[131] Weisong SHI, Jie CAO, Quan ZHANG, Youhuizi LI et Lanyu XU. "Edge Computing : Vision and Challenges". In : *IEEE Internet of Things Journal* 3.5 (2016), p. 637-646. DOI : 10.1109/JIOT.2016.2579198.

[132] Camélia SLIMANI, Stéphane RUBINI et Jalil BOUKHOBZA. "Hymad : a hybrid memory-aware DVFS strategy". In : *ACM SIGBED Review* 16.3 (2019), p. 45-50.

[133] Camélia SLIMANI, Stéphane RUBINI et Jalil BOUKHOBZA. "K -MLIO : Enabling K -Means for Large Data-Sets and Memory Constrained Embedded Systems". In : *27th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2019, Rennes, France, October 21-25, 2019*. 2019, p. 262-268. DOI : 10.1109/MASCOTS.2019.00037. URL : https://doi.org/10.1109/MASCOTS.2019.00037.

[134] Camélia SLIMANI, Chun-Feng WU, Yuan-Hao CHANG, Stéphane RUBINI et Jalil BOUKHOBZA. "RaFIO : A Random Forest I/O-Aware algorithm". In : *The 36th ACM Symposium on Applied Computing (SAC 2021)*. 2021.

[135] Linghao SONG, Xuehai QIAN, Hai LI et Yiran CHEN. "PipeLayer : A Pipelined ReRAM-Based Accelerator for Deep Learning". In : *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2017, p. 541-552. DOI : 10.1109/HPCA.2017.55.

[136] Nikola STOJANOVIĆ, Vladan DAMJANOVIĆ et Srđan VUKMIROVIĆ. "Parking Occupancy Prediction using Computer Vision with Location Awareness". In : *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*. IEEE. 2021, p. 1-5.

[137] Pan Jun SUN. "Privacy Protection and Data Security in Cloud Computing : A Survey, Challenges, and Solutions". In : *IEEE Access* 7 (2019), p. 147420-147452. DOI : 10.1109/ACCESS.2019.2946185.

[138] Zahir TARI. "Security and Privacy in Cloud Computing". In : *IEEE Cloud Computing* 1.01 (2014), p. 54-57. ISSN : 2372-2568. DOI : 10.1109/MCC.2014.20.

[139] *Tensorflow Lite : Ml pour appareils mobiles et de Périphérie*. URL : https://www.tensorflow.org/lite?hl=fr.

[140] Abdessamad TIOUIOUINE, Suzanne YAMEOGO, Vincent VALLES, Laurent BARBIERO, Fabrice DASSONVILLE, Marc MOULIN, Tarik BOURAMTANE, Tarik BAHAJ, Moad MORARECH et Ilias KACIMI. "Dimension Reduction and Analysis of a 10-Year Physicochemical and Biological Water Database Applied to Water Resources Intended for Human Consumption in the Provence-Alpes-Côte d'Azur Re-

gion, France". In : *Water* 12.2 (2020). ISSN : 2073-4441. DOI : 10.3390/w12020525. URL : https://www.mdpi.com/2073-4441/12/2/525.

[141] Hai Nam TRAN. "Cache memory aware priority assignment and scheduling simulation of real-time embedded systems". Theses. Université de Bretagne occidentale - Brest, jan. 2017. URL : https://tel.archives-ouvertes.fr/tel-01773862.

[142] Hristos TYRALIS, Georgia PAPACHARALAMPOUS et Andreas LANGOUSIS. "A Brief Review of Random Forests for Water Scientists and Practitioners and Their Recent History in Water Resources". In : *Water* 11.5 (2019). ISSN : 2073-4441. DOI : 10.3390/w11050910. URL : https://www.mdpi.com/2073-4441/11/5/910.

[143] Olivier VALERY, Pangfeng LIU et Jan-Jan WU. "CPU/GPU Collaboration Techniques for Transfer Learning on Mobile Devices". In : *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. 2017, p. 477-484. DOI : 10.1109/ICPADS.2017.00069.

[144] Olivier VALERY, Pangfeng LIU et Jan-Jan WU. "Low Precision Deep Learning Training on Mobile Heterogeneous Platform". In : *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. 2018, p. 109-117. DOI : 10.1109/PDP2018.2018.00023.

[145] Andrea VATTANI. "K-means Requires Exponentially Many Iterations Even in the Plane". In : *Discrete & Computational Geometry* 45.4 (2011), p. 596-616. ISSN : 1432-0444. DOI : 10.1007/s00454-011-9340-1. URL : https://doi.org/10.1007/s00454-011-9340-1.

[146] Swagath VENKATARAMANI, Ashish RANJAN, Subarno BANERJEE, Dipankar DAS, Sasikanth AVANCHA, Ashok JAGANNATHAN, Ajaya DURG, Dheemanth NAGARAJ, Bharat KAUL, Pradeep DUBEY et Anand RAGHUNATHAN. "SCALEDEEP : A scalable compute architecture for learning and evaluating deep networks". In : *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 2017, p. 13-26. DOI : 10.1145/3079856.3080244.

[147] Paul VOIGT et Axel von dem BUSSCHE. *The EU General Data Protection Regulation (GDPR) : A Practical Guide*. 1st. Springer Publishing Company, Incorporated, 2017. ISBN : 3319579584.

[148] Hadley WICKHAM. "Tidy Data". In : *Journal of Statistical Software, Articles* 59.10 (2014), p. 1-23. ISSN : 1548-7660. DOI : 10.18637/jss.v059.i10. URL : https://www.jstatsoft.org/v059/i10.

[149] Marvin N WRIGHT et Andreas ZIEGLER. "ranger : A Fast Implementation of Random Forests for High Dimensional Data in C++ and R". In : *Journal of Statistical Software, Articles* 77 (2017).

[150] Nan WU et Yuan XIE. "A Survey of Machine Learning for Computer Architecture and Systems". In : *ArXiv* abs/2102.07952 (2021).

[151] Xindong WU, Vipin KUMAR, Ross QUINLAN, Joydeep GHOSH, Qiang YANG, Hiroshi MOTODA, G. MCLACHLAN, Shu Kay Angus NG, Bing LIU, Philip YU, Zhi-Hua ZHOU, Michael STEINBACH, David HAND et Dan STEINBERG. "Top 10 algorithms in data mining". In : *Knowledge and Information Systems* 14 (déc. 2007). DOI : 10.1007/s10115-007-0114-2.

[152] Chen XIN, Qiang CHEN, Miren TIAN, Mohan JI, Chenglong ZOU, Xin'An WANG et Bo WANG. "COSY : An Energy-Efficient Hardware Architecture for Deep Convolutional Neural Networks Based on Systolic Array". In : *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. 2017, p. 180-189. DOI : 10.1109/ICPADS.2017.00034.

[153] Dianlei XU, Tong LI, Yong LI, Xiang SU, Sasu TARKOMA, Tao JIANG, Jon CROWCROFT et Pan HUI. "Edge Intelligence : Architectures, Challenges, and Applications". In : *arXiv e-prints* (2020), arXiv-2003.

[154] Dongkuan XU et Ying jie TIAN. "A Comprehensive Survey of Clustering Algorithms". In : *Annals of Data Science* 2 (2015), p. 165-193.

[155] M-S YANG. "A survey of fuzzy clustering". In : *Mathematical and Computer modelling* 18.11 (1993), p. 1-16.

[156] Tao YANG, Qiang REN, Fangbing ZHANG, Bolin XIE, Hailei REN, Jing LI et Yanning ZHANG. "Hybrid camera array-based uav auto-landing on moving ugv in gps-denied environment". In : *Remote Sensing* 10.11 (2018), p. 1829.

[157] Mahmut Taha YAZICI, Shadi BASURRA et Mohamed Medhat GABER. "Edge Machine Learning : Enabling Smart Internet of Things Applications". In : *Big Data and Cognitive Computing* 2.3 (2018). ISSN : 2504-2289. DOI : 10.3390/bdcc2030026. URL : https://www.mdpi.com/2504-2289/2/3/26.

[158] Cha ZHANG et Yunqian MA. *Ensemble machine learning : methods and applications*. Springer, 2012.

[159] Chen ZHANG, Peng LI, Guangyu SUN, Yijin GUAN, Bingjun XIAO et Jason CONG. "Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks". In : *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '15. Monterey, California, USA : Association for Computing Machinery, 2015, 161–170. ISBN : 9781450333153. DOI : 10.1145/2684746.2689060. URL : https://doi.org/10.1145/2684746.2689060.

[160] Tian ZHANG, Raghu RAMAKRISHNAN et Miron LIVNY. "BIRCH : An Efficient Data Clustering Method for Very Large Databases". In : *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. SIGMOD '96. Montreal, Quebec, Canada : Association for Computing Machinery, 1996, 103–114. ISBN : 0897917944. DOI : 10.1145/233269.233324. URL : https://doi.org/10.1145/233269.233324.

[161] Qihua ZHOU, Zhihao QU, Song GUO, Boyuan LUO, Jingcai GUO, Zhenda XU et Rajendra AKERKAR. "On-Device Learning Systems for Edge Intelligence : A Software and Hardware Synergy Perspective". In : *IEEE Internet of Things Journal* 8.15 (2021), p. 11916-11934. DOI : 10.1109/JIOT.2021.3063147.

[162] Zhi ZHOU, Xu CHEN, En LI, Liekang ZENG, Ke LUO et Junshan ZHANG. "Edge Intelligence : Paving the Last Mile of Artificial Intelligence With Edge Computing". In : *Proceedings of the IEEE* 107.8 (2019), p. 1738-1762. DOI : 10.1109/JPROC.2019.2918951.

[163] Hangyu ZHU, Jinjin XU, Shiqing LIU et Yaochu JIN. "Federated learning on non-IID data : A survey". In : *Neurocomputing* 465 (2021), p. 371-390. ISSN : 0925-2312. DOI : https://doi.org/10.1016/j.neucom.2021.07.098. URL : https://www.sciencedirect.com/science/article/pii/S0925231221013254.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

Université de Bretagne Occidentale

Titre : Vers des algorithmes d'apprentissage automatique économes en E/S pour les systèmes embarqués : application aux K-means et Random Forests

Mots clés : Edge Intelligence, Machine Learning, contrainte mémoire, E/S, K-means, Random Forest

Résumé : L'abondance des données numériques collectées permet d'alimenter des modèles intelligents, capables d'extraire de la connaissance exploitable par l'humain. L'entraînement de ces modèles, par des algorithmes de Machine Learning, requiert d'importantes ressources de calcul, de mémoire et de stockage. Par conséquent, cette tâche est effectuée sur des calculateurs puissants, ce qui nécessite le transfert des données des dispositifs de collecte, vers les calculateurs. Pour s'affranchir des coûts de communications, le paradigme de l'Edge Intelligence (EI) émerge, et rapproche l'intelligence au plus près des dispositifs de collecte. L'EI soulève de nombreux défis scientifiques, notamment l'exécution des algorithmes de ML sous des contraintes temporelles, énergétiques, et d'espace mémoire, présentes sur les dispositifs embarqués. Nous nous intéressons particulièrement à la contrainte d'espace en mémoire principale. Cet espace sur les dispositifs de collecte est limité et peut être insuffisant pour contenir les données d'apprentissage. Lorsque le volume de données à traiter est supérieur à l'espace mémoire disponible, le système d'exploitation utilise le mécanisme de swap afin d'étendre l'espace mémoire disponible par une partie du stockage secondaire. Par conséquent, l'exécution de la phase d'apprentissage est ralentie par les accès au stockage secondaire (E/S), bien plus lents que ceux à la mémoire principale. Notre objectif est d'analyser le motif des E/S des algorithmes de ML, de déterminer son origine, et de proposer une méthode de réduction des E/S pour accélérer ces algorithmes sur des plateformes embarquées. Nous nous sommes intéressés à deux cas d'études algorithmiques qui sont : le K-MEANS et les RANDOM FORESTS. En résumé, nous avons montré à travers deux exemples, que la révision des algorithmes permet de faire face à l'augmentation du volume de données d'apprentissage en réduisant le volume des E/S. L'extension de cette démarche à d'autres algorithmes/familles d'algorithmes de ML pourrait faire l'objet de travaux futurs.

Title: Towards I/O-efficient Edge Intelligence algorithms: K-means and Random Forests

Keywords: Edge Intelligence, Machine Learning, memory constraint,

Abstract: The abundance of digital data collected makes it possible to feed intelligent models, capable of extracting knowledge that can be used by humans. The training of these models, by Machine Learning algorithms, requires important computing, memory and storage resources. Consequently, this task is performed on powerful computers, which requires the transfer of data from the collection devices to the computers. To overcome the communication costs, the Edge Intelligence (EI) paradigm is emerging, bringing intelligence closer to the collection devices. EI raises many scientific challenges, including the execution of ML algorithms under time, energy, and memory space constraints present on embedded devices. We are particularly interested in the main memory space constraint. This space on collection devices is limited and may be insufficient to hold the training data. When the amount of data to be processed is larger than the available memory space, the operating system uses the swap mechanism to extend the available memory space by some of the secondary storage. As a result, the execution of the learning phase is slowed down by the accesses to the secondary storage (I/O), which are much slower than those to the main memory. Our goal is to analyze the I/O pattern of ML algorithms, determine its origin, and propose an I/O reduction method to accelerate these algorithms on embedded platforms. We have focused on two algorithmic case studies which are: K-MEANS and RANDOM FORESTS. In summary, we have shown through two examples, that the revision of algorithms can cope with the increase in the volume of training data by reducing the volume of I/O. The extension of this approach to other ML algorithms/families could be the subject of future work.