
TX00 "Green AI"

Guide pour les étudiants ingénieurs et informaticiens en Green Computing

Julie GUILLERMET & Rosalie JARDRI

GI02



Semestre : P24

Remerciements

Nous tenons à remercier le professeur Marc Shawky de nous avoir guidées durant nos recherches ainsi que pour ses conseils et l'aide apportée lors de la réalisation de notre TX durant le semestre P24.

Nous souhaitons également remercier l'équipe de FinalSpark, en particulier Ewelina Kurtys et Jean Marc Comby, pour avoir gentiment répondu à nos sollicitations lors de notre tentative de test de nos algorithmes sur leur bioprocasseur et pour avoir partagé avec nous des ressources bibliographiques utiles.

Enfin, nous souhaitons remercier Floryan De Vuyst pour avoir répondu si rapidement à nos questions concernant le nouveau supercalculateur NVIDIA de l'UTC, même si sa date de mise en service ne nous a malheureusement pas permis de l'utiliser pour ce rapport.

Table des matières

Introduction	1
1 - Consommation énergétique du hardware au software	3
1.1 Consommation énergétique des infrastructures et des outils de programmation	3
1.1.1 Les data centers et les architectures	3
1.1.2 Les langages de programmation	11
1.2 Consommation énergétique de l'intelligence artificielle	14
1.2.1 Cas du Machine Learning	15
1.2.2 Cas du Deep Learning	19
2 - Techniques de Green Computing	24
2.1 Pour réduire la consommation d'énergie de l'intelligence artificielle	24
2.1.1 Au sein des algorithmes de Machine Learning	24
2.1.2 Au sein des algorithmes de Deep Learning	28
2.2 Pour réduire la consommation d'énergie au sein des architectures et infrastructures	35
2.2.1 Au sein des architectures	35
2.2.2 Dans les data centers	39
3 - Critiques	45
Conclusion	48
Références bibliographiques	49
Annexe 1. Liste des figures et des tableaux du rapport	i
Annexe 2. KNN avec 3 voisins	iv
Annexe 3. KMeans avec 256 clusters	vi
Annexe 4. Fourier - génération d'image	x
Annexe 5. KMeans - génération d'image	xii
Annexe 6. Perceptron multicouches avec 10 couches cachées	xiv
Annexe 7. Réseau de neurones récurrents avec 2 couches cachées	xvi
Annexe 8. Réseau neuronal convolutif avec 7 couches cachées	xviii
Annexe 9. Réseau antagoniste génératif	xx

Introduction et concepts

Aujourd'hui, deux grands sujets de société captent régulièrement l'attention du public : l'essor de l'informatique et le changement climatique. Ces thématiques sont pourtant souvent envisagées de manière indépendante, les progrès technologiques semblant éloignés des préoccupations liées au réchauffement climatique. La question de la consommation d'énergie dans le domaine du numérique et plus particulièrement de *l'intelligence artificielle* (IA) est pourtant cruciale. Pour s'en convaincre, il suffit de constater que *"l'empreinte carbone de la formation d'un seul grand modèle linguistique équivaut à environ 300 000 kg d'émissions de dioxyde de carbone. Ce chiffre est de l'ordre de 125 vols aller-retour entre New York et Pékin, une quantification que les profanes peuvent visualiser"* (Dhar [2020]). Il existe donc un réel besoin de quantification de ces émissions.

En effet, le numérique et plus particulièrement l'informatique, est un domaine qui, par son évolution constante et son impact croissant sur le quotidien, nécessite une quantité considérable d'énergie. En 2022, d'après le GIEC, la part des émissions de gaz à effet de serre dans les émissions mondiales atteignait 3%. Ces nouveaux usages ont nécessité la création de centres de données gigantesques, alimentés en continu, qui hébergent des milliards d'informations et supportent le fonctionnement de nos applications, sites web et services en ligne. Cette demande énergétique toujours plus grande est souvent occultée derrière l'illusion de l'immédiateté et de la fluidité que nous offre le monde numérique. Pourtant, les serveurs, les dispositifs de stockage et les infrastructures de télécommunications qui soutiennent cet univers virtuel ont un impact environnemental significatif, contribuant aux émissions de gaz à effet de serre et à l'empreinte carbone globale (Arcep [2023]).

L'IA d'autre part, véritable révolution technologique de ces dernières années, repose sur des algorithmes complexes et des modèles d'apprentissage automatique (i.e., *Machine-Learning* ou ML) et exige des ressources informatiques considérables. Il est important ici de ne pas confondre apprentissage automatique et apprentissage profond (i.e., *Deep-Learning* ou DL). En effet, les algorithmes de DL constituent en réalité une sous-branche du ML, analysant des modèles et/ou des relations entre données en s'appuyant sur des réseaux de neurones (et non plus sur des modèles statistiques comme en ML). Le ML ou les réseaux de neurones convolutifs, type particulier de DL (pour ne citer qu'eux) induisent des calculs intensifs responsables d'une très importante consommation d'énergie. Les serveurs qui exécutent ces tâches sont souvent exploités à pleine capacité, ce qui aggrave encore plus leur empreinte environnementale.

Face à cette réalité, de plus en plus d'articles sont publiés pour sensibiliser, conseiller ou encore alerter sur l'accroissement des besoins énergétiques engendrés par le numérique. Les entreprises technologiques commencent également à prendre conscience de leur responsabilité environnementale et s'engagent à réduire leur empreinte carbone. Des initiatives visant à optimiser les algorithmes utilisés, à développer des architectures matérielles moins gourmandes en énergie et à recourir aux énergies renouvelables pour alimenter les centres de données émergent progressivement.

Il est intéressant de noter que l'unité de mesure généralement utilisée pour quantifier et comparer les émissions de carbone est celle des **équivalents CO₂**. Cette unité nous permet de comparer différentes sources d'émissions de *gaz à effet de serre* (GES¹) en utilisant un dénominateur commun, celui des grammes de CO₂ émis par kilowatt-heure d'électricité produite (gCO₂eq/kWh). Le CO₂ sert alors de "norme" : son potentiel de réchauffement global est fixé arbitrairement à 1. La consommation énergétique est, quant à elle, quantifiée en Joules ou KiloWatt-Heure.

Toutes ces réflexions nous amènent à nous poser les questions suivantes : Quels sont les postes énergétiques les plus importants dans le domaine du numérique ? Comment analyser efficacement l'empreinte carbone des modèles d'intelligence artificielle ? Et quels sont les leviers d'action et les bonnes pratiques qui en découlent ?

Pour répondre à cette problématique, nous nous intéresserons dans un premier temps à la consommation énergétique du *hardware* (i.e., des infrastructures et équipements nécessaires pour faire fonctionner les algorithmes d'IA) et du *software* (les algorithmes eux-mêmes). Une fois les coûts identifiés, nous nous intéresserons aux outils permettant de traquer ces émissions carbone et aux techniques de *Green Computing* permettant de les réduire, et ce, encore une fois, au sein des algorithmes d'IA (ML et DL) et via des architectures spécifiques.

★ ★ ★

1. Les 6 GES suivis dans le cadre du protocole de Tokyo sont : le dioxyde de carbone (CO₂), le méthane (CH₄), le protoxyde d'azote (N₂O), l'hexafluorure de soufre (SF₆), les hydrofluorocarbures (HFC) et les perfluorocarbures (PFC). <https://www.foresteam.fr/post/equivalent-co2>

1 - Consommation énergétique du hardware au software

1.1 Consommation énergétique des infrastructures et des outils de programmation

Pour commencer, nous étudierons la consommation énergétique des infrastructures et des outils informatiques de manière globale. Nous avons choisi de nous intéresser particulièrement à trois niveaux d'observation et de mesure. Premièrement, notre analyse portera sur les centres de données, ou *data centers*, qui consomment une quantité massive d'énergie pour stocker, traiter et distribuer des données. En parallèle, nous regarderons de plus près le fonctionnement des ordinateurs ou des calculateurs manipulant ces données (programmes, calculs, etc.). Finalement, nous détaillerons en quoi le choix des langages de programmation utilisés pour développer les logiciels influence également l'efficacité énergétique des applications. Cette section explore ces aspects critiques en détaillant la consommation énergétique des *data centers*, des ordinateurs/calculateurs et l'impact du langage de programmation choisi.

1.1.1 Les data centers et les architectures

L'informatique nuagique, ou *cloud computing*, désigne un système donnant accès à des services informatiques, comme le stockage ou le calcul, ainsi qu'à des logiciels, par le biais des réseaux internet ou de réseaux privés. Il repose sur des infrastructures physiques : les **data centers**, également connus sous le nom de centres de données. Ce sont des installations cruciales pour le stockage, la gestion et la distribution de données. Ils consomment une quantité significative d'énergie en raison de l'alimentation continue des serveurs, des systèmes de refroidissement et de la gestion des infrastructures de réseau. Selon certaines estimations, les data centers représentent environ 1% de la consommation électrique mondiale, et cette proportion est en constante augmentation avec la croissance exponentielle des données numériques (Grégory Lebourg [2024]).

Les data centers sont constitués des éléments suivants (cf. Figure 1) :

- (1) **salle serveur**. Les serveurs sont hébergés dans des baies verticales (dont la capacité est exprimée en terme de tension). Ces baies sont empilées les unes à côté des autres. La chaleur est évacuée à l'arrière du dispositif, avec des allées chaudes (où la chaleur est évacuée) et des allées froides (où le personnel peut se rendre en cas de problème).

- (2) **salle énergie**. Elle contient des transformateurs, jouant un rôle crucial dans l'alimentation en énergie, puisqu'ils peuvent convertir la haute-tension en tension uti-

lisible par les équipements informatiques, et distribuer l'énergie aux composants. Ces transformateurs sont refroidis à l'huile et sont donc souvent situés à l'extérieur du bâtiment pour limiter les risques d'incendie. Ils sont dits secs, c-à-d. qu'ils fonctionnent très bien sans maintenance. Ainsi, l'UPS (ou *alimentation sans interruption* en français) joue un rôle central en assurant une alimentation continue et stable, transformant le courant continu des batteries en courant alternatif pour les serveurs, et en permettant la transition vers les générateurs de secours en cas de coupure prolongée.

- (3) **toitures.** Ces dernières sont aérothermes. Elles peuvent renvoyer une grande partie de la chaleur solaire (via des matériaux réfléchissants) et sont de très bons isolants thermiques. Elles jouent un rôle crucial dans la gestion de l'énergie thermique.

- (4) **cuves de stockage de fuel.** Elles garantissent la sauvegarde (un *backup*) en cas de panne d'électricité, si le système doit être alimenté en énergie de toute urgence. Les data centers sont normalement équipés pour fonctionner en autonomie énergétique sur une période d'au moins 48h.

- (5) **bureaux pour le personnel.**

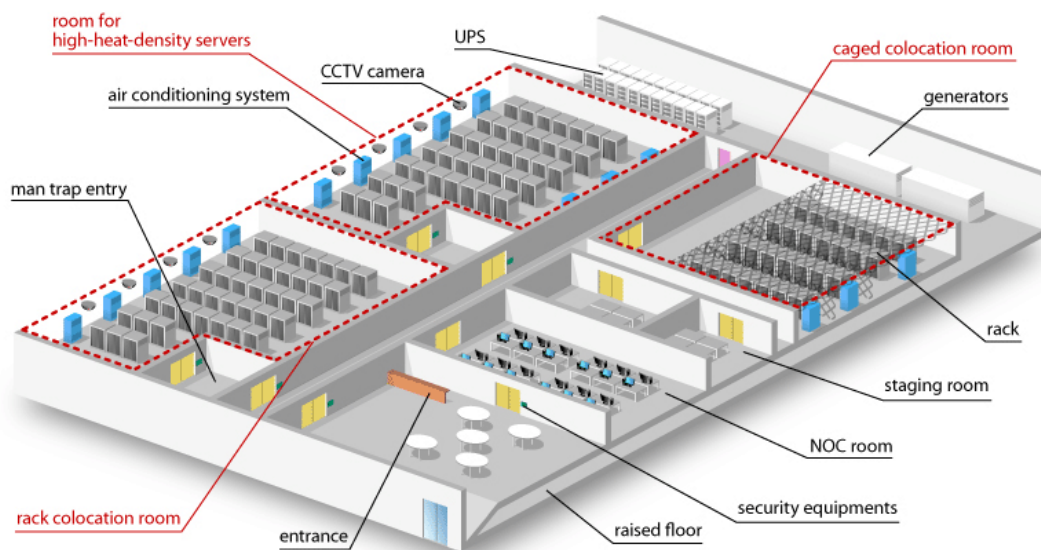


FIGURE 1 – Architecture typique d'un *data center*. Source : PMP, GreenFlex.

Une des composantes physiques majeure des data centers est le système de refroidissement (Grégory Lebourg [2024]). Il existe 3 grands types de systèmes de refroidissement (cf. Figure 2) : (i) le *direct free-cooling*, (ii) l'*indirect free-cooling* et (iii) le *refroidissement mécanique*.

Le **direct free-cooling** permet de récupérer de l'air frais à l'extérieur du bâtiment (qui est alors non isolé) puis d'y rejeter de l'air chaud. Il assure seulement le flux d'air d'un point de vue mécanique. Il s'agit de la meilleure solution actuellement. Sa principale limite est la nécessité de pouvoir accéder à de l'air ambiant frais (plus simple dans des pays comme l'Islande ou la Finlande qu'en Espagne par exemple). En France, lorsque la température de l'air ambiant devient trop élevée pour du direct free-cooling, on fait passer l'air dans des circuits réfrigérés pour le refroidir. Il existe également des data centers immergés. Le principal problème est qu'il est alors très difficile d'y accéder pour changer des pièces.

L'**indirect free-cooling** permet de capter les calories à l'intérieur du bâtiment (ici, on suppose que le bâtiment est bien isolé et étanche). Un fluide froid rentre dans le système et ressort du circuit chaud, après échange thermique. Ce système utilise pour cela un fluide caloporteur intermédiaire.

Le **refroidissement mécanique** est la méthode la plus répandue (e.g., c'est le même principe utilisé dans les réfrigérateurs au niveau des compresseurs). Elle utilise beaucoup d'électricité et nécessite un moteur. Quand le milieu est humide, le système est obligatoirement basé sur un refroidissement mécanique. C'est malheureusement le moins efficace et le plus énergivore.

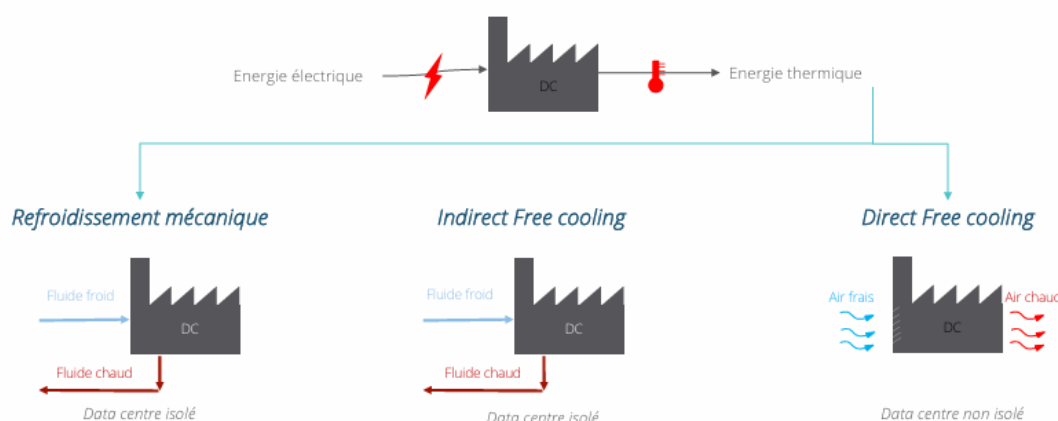


FIGURE 2 – Grandes familles de systèmes de refroidissements (Grégory Lebourg [2024]).

Une étude récente montre que *"c'est la fabrication de terminaux et des infrastructures de réseaux qui pèse le plus lourd dans le bilan carbone du numérique mondial, suivie par la consommation des équipements, du réseau et des fermes de serveurs (data centers). La construction d'un ordinateur portable émet ainsi environ 330 kilogrammes d'équivalent CO₂, tout en nécessitant énormément d'eau et de matières premières, notamment des métaux comme le palladium, le cobalt ou les terres rares. Le fonctionnement des data centers génère à lui seul 19 % de l'empreinte énergétique totale du numérique"* (Sébastien Broca [2020]).

En effet, "30 à 50 % de l'énergie totale consommée par les centres de données correspond à la consommation d'énergie de refroidissement gaspillée par des systèmes de refroidissement inefficaces" (Uddin et al. [2015]). En outre, les principaux impacts des data centers sont au nombre de 4 : (i) le changement climatique, (ii) le stress hydrique, (iii) l'artificialisation des sols, et (iv) l'épuisement des ressources abiotiques (Shehabi et al. [2011]).

Au-delà de ces infrastructures, les ordinateurs en eux-mêmes consomment énormément. Mais comment fonctionnent ces derniers et où se situent ces coûts ?

Les principaux composants d'un ordinateur (Figure 3) sont la (i) *mémoire* permettant le stockage des informations, (ii) le *processeur* manipulant les informations et permettant ainsi leur traitement en effectuant par exemple des calculs et en renvoyant un résultat et (iii) les *entrées/sorties* permettant la communication avec l'utilisateur, ou l'extérieur en général, en codant l'information en entrée et en la décodant en sortie. Enfin, il existe des *bus* (i.e., dispositifs reliant ces composants et leur permettant de communiquer). Le *micro-contrôleur* rassemble tous ces éléments dans un circuit intégré².

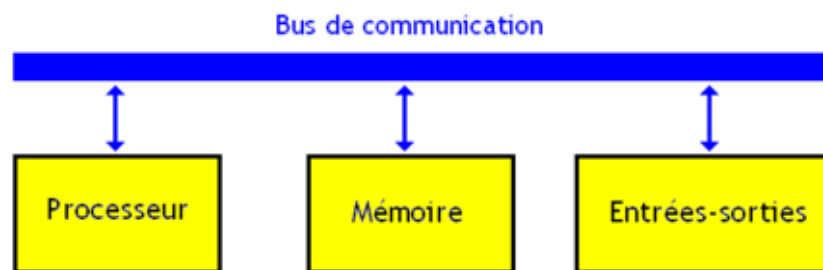


FIGURE 3 – Architecture minimale d'un ordinateur. Source "Fonctionnement d'un ordinateur".

La mémoire d'un micro-contrôleur est découpée en **bytes**, des blocs mémoires possédant « *un nombre fini et constant de bits* » (i.e. 8 bits, ce qui correspond à un octet). Le nombre de bits qu'une mémoire peut contenir correspond à sa capacité maximale. Il existe deux types d'actions l'**écriture** et la **lecture** et deux types de mémoires : la mémoire ROM (*Read Only Memory*), qui peut uniquement lire et récupérer des données sans les modifier, et la mémoire RWN (*Read Write Memory*), qui peut faire les deux (cf. Figure 4). Plus précisément, la ROM est utilisée pour stocker des programmes et des constantes (ses informations sont lues par le processeur), tandis que la RWN est utilisée pour stocker les résultats des calculs (i.e., des données temporaires), qui sont cependant nécessaires pour que les programmes stockés dans la ROM fonctionnent. Tous les ordinateurs possèdent ces deux types de mémoire. Lors de l'adressage (i.e. sélection du byte auquel le processeur veut accéder pour effectuer la lecture/écriture), un byte est limité à une action³.

2. Pour la partie sur le fonctionnement de l'ordinateur, nous nous sommes principalement basée sur le wiki-livre intitulé : "Fonctionnement d'un ordinateur".

3. L'adresse, qui permet l'adressage, correspond ici à un numéro spécifique du byte.

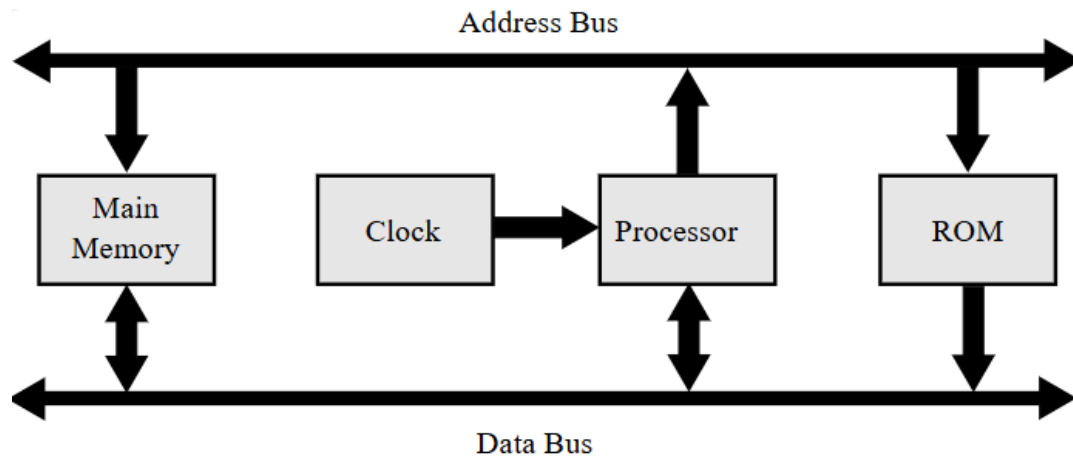


FIGURE 4 – Architecture avec une ROM et une RWM. Source : livre Fonctionnement d'un ordinateur.

Le processeur, aussi appelé *Central Processing Unit* ou CPU, est donc un circuit effectuant les calculs, manipulant l'information venant de l'entrée/sortie ou récupérée dans la mémoire. Il y a un nombre limité d'opérations que le CPU peut effectuer : (i) les **instructions arithmétiques** (addition, soustraction, division, multiplication), (ii) les **instructions de test** (comparer deux nombres entre eux puis agir en fonction), (iii) les **instructions d'accès mémoire** (échanger de données entre mémoire et processeur). Un programme correspond ainsi simplement à une suite d'instructions.

Tous les ordinateurs ont au moins un CPU. En effet, la grande majorité des ordinateurs sont mono-processeurs mais certains ordinateurs peuvent en avoir plusieurs (on parle alors de multi-processeurs, i.e., plusieurs CPU sur la même carte mère). En outre, deux CPU permettent de faire deux fois plus de calcul et ainsi de gagner en performance. C'est donc très commun pour les supercalculateurs⁴ mais aussi sur les serveurs. Il existe divers moyens pour utiliser plusieurs CPU en même temps : (i) exécuter des programmes différents sur des processeurs différents, (ii) utiliser des programmes adaptés de répartition du calcul sur plusieurs CPU, ou (iii) utiliser le même programme sur différents CPU, mais traiter un ensemble de données différent. Aujourd'hui, le plus souvent, aucune de ces solutions n'est effective : à la place, les ordinateurs possèdent des CPU multi-cœurs, c'est-à-dire qu'un seul CPU est présent sur la carte-mère mais avec plusieurs puces similaires dans le même boîtier. On dit alors qu'il y a plusieurs cœurs⁵ exécutant chacun un programme différent.

Le clavier, la souris ou encore la carte son sont des exemples d'entrées : ils convertissent les informations en codes binaires, qui circulent ensuite dans les bus. C'est ce qu'on

4. Un supercalculateur est un très grand ordinateur, réunissant plusieurs dizaines de milliers de processeurs, capable de réaliser un très grand nombre d'opérations de calcul simultanées. (déf. CEA)

5. Un cœur est ensemble de circuits nécessaires pour exécuter un programme. Il y a des processeurs double-cœur, quadruple-cœur, octuple-cœur (devenu la norme pour les particuliers, les logiciels et systèmes d'exploitations).

appelle l'encodage. L'écran LCD, l'imprimante et la carte son⁶ sont des exemples de sortie : ils transforment les codes binaires en informations perceptibles ou compréhensibles. C'est ce qu'on appelle le décodage.

Notons que les périphériques sont différents des entrées/sorties. En effet, les périphériques sont connectés à l'unité centrale (par exemple, les : claviers, souris, webcam, imprimantes, clés USB, disques durs externes, câbles Ethernet de la Box Internet, etc.). Alors que les entrées/sorties contiennent les périphériques, mais aussi d'autres composants, comme les cartes d'extensions (carte son, carte graphique), ou certains composants de la carte-mère. Toutes les entrées/sorties contiennent des registres d'interfaçage. Ils agissent en tant qu'intermédiaire entre les périphériques et le reste de l'ordinateur. Ils peuvent être variés : (i) registres de données (échange entre le CPU et le périphérique en question), (ii) registre de lecture ou d'écriture (parfois fusionnés), (iii) registre d'état (lisible par le CPU, qui donne l'état du périphérique, mais n'est pas toujours disponible).

Les bus de communication (cf. Figure 5) relient le CPU avec la mémoire et les entrées/sorties. Comme expliqué précédemment, c'est un ensemble de lignes électriques, qui reçoivent des codes binaires (des 0 et des 1). Il y a en toujours au moins un par ordinateur et ils doivent remplir trois pré-requis : (i) être capable de sélectionner la mémoire (ou entrée/sortie) dont on a besoin ; (ii) être capable de transmettre le type d'action (une lecture ou une écriture), et (iii) être capable de transférer de la donnée. De ces trois pré-requis, on peut distinguer trois types de bus distincts : les *bus de données*, qui permettent d'échanger les données entre composants, les *bus de commande*, qui permettent au CPU de configurer la mémoire et les entrées/sorties, et les *bus d'adresse*, qui ne sont pas obligatoires et permettent au CPU de sélectionner l'entrée/sortie ou portion de mémoire qui l'intéresse.

Enfin, au sein des ordinateurs, les données et les instructions sont normalement séparées. Ainsi, les données sont en générales stockées dans la mémoire travail, ou RAM (*Random Access Memory* ou mémoire vive), et les instructions sont stockées dans la mémoire programme, ou ROM,⁷ Pour certaines architectures, cette séparation est nette, c'est le cas des architectures **Harvard**, avec des accès parallèles (i.e. accès simultanés aux instructions et données à l'aide de bus séparés). Pour d'autres, cette séparation est moins franche, c'est le cas des architectures Von Neumann (où les données et instructions sont traitées par le CPU). De plus, les instructions sont exécutées de manière séquentielle (i.e., une à la fois pour un bus donné). Cette dernière architecture est par exemple utilisée dans les ordinateurs portables. En effet, les architectures Von Neumann sont plus coûteuses⁸ que les architectures Harvard (qui ont en général une meilleure performance mais sont plus complexes).

6. La carte son est à la fois une entrée et une sortie.

7. Il existe une distinction entre la **mémoire travail** (qui mémorise les variables utiles au bon fonctionnement des programmes) et la **mémoire programme** (qui stocke les programmes en question attendant d'être exécutés).

8. Puisqu'elles nécessitent un transfert constant des données entre CPU et mémoire, les deux étant séparés.

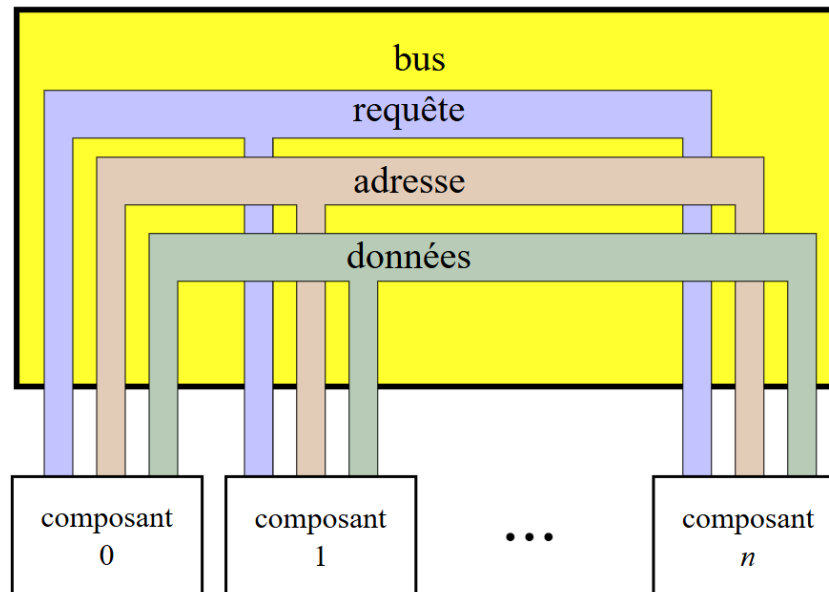


FIGURE 5 – Contenu d'un bus informatique. Source : Fonctionnement d'un ordinateur.

Il existe d'autres types d'architecture informatique, comme par exemple les architectures neuromorphiques, qui permettent de traiter massivement des données en parallèle et fonctionnent de manière asynchrone (i.e., il n'y a pas « d'horloge globale » pour synchroniser les opérations⁹), contrairement aux architectures Von Neumann et Harvard, qui sont synchrones.

Intéressons-nous maintenant à la performance d'un ordinateur : De quoi dépend-elle ? Quel lien peut-on faire entre performance et consommation énergétique ?

Il est tout d'abord important de noter qu'on ne peut pas comparer la performance d'un processeur, avec celle d'une mémoire¹⁰ ou d'une entrée/sortie. Nous nous attarderons ici principalement sur la performance du CPU. En effet, le CPU peut être comparé au « cerveau » de l'ordinateur en tant qu'unité principale. En outre, la performance et la consommation énergétique du CPU jouent un rôle central dans la détermination de la performance globale du système informatique. « *Plus la fréquence du processeur est élevée, plus il est puissant* ».

On peut commencer par s'intéresser au temps d'exécution d'un programme. Ce temps correspond au produit entre le nombre d'instructions (N) et la durée moyenne d'une instruction (en seconde), soit : $T = N \times T_{instruction}$.

9. Ces dernières sont déclenchées par la disponibilité des données ou des événements locaux, ce qui permet d'avoir une dissipation thermique plus faible et une réduction de la latence soit une meilleure consommation énergétique. Les architectures neuromorphiques et leurs intérêts seront détaillées plus tard dans ce rapport.

10. Par exemple, la performance d'une mémoire se calcule grâce au temps nécessaire pour effectuer une lecture/écriture et au débit mémoire ou binaire, soit la quantité d'informations enregistrées en un temps donné (une seconde). Plus ce temps est bas et plus le débit est élevé, plus la mémoire est rapide.

Cette formule peut également s'écrire comme ceci : $T = N \times \frac{CPI}{f} = \frac{N}{IPC \times f}$, où f correspond à la fréquence, CPI , au nombre de cycles d'horloge par instruction et IPC , le nombre de calculs effectués par cycle.

En effet, il n'est pas possible de ne prendre en compte que la fréquence lorsque l'on compare deux processeurs. Il faut également se baser sur l'IPC (*Instruction Per Cycle*), qui n'est en général pas mentionnée par les fabricants, contrairement à la fréquence¹¹. Ainsi, si on regarde l'équation précédente, on peut voir qu'il existe trois moyens d'augmenter la vitesse d'exécution : (i) diminuer N , (ii) augmenter f et (iii) diminuer le CPI (ou augmenter l'IPC).

Au fil du temps, les processeurs sont devenus de plus en plus performants : *"Les raisons à cela sont doubles, mais liées aux lois de Dennard. La première raison est la hausse de la fréquence. C'était une source importante avant 2005, avant que les lois de Dennard cessent de fonctionner. L'autre raison, est la loi de Moore."*

En effet, les premières stratégies augmentaient principalement la fréquence, et les CPU se sont mis à fonctionner à très haute fréquence (par exemple à l'aide de très long pipelines). Cette hausse de la fréquence était cependant problématique puisqu'elle induisait une surchauffe du CPU (« *dissipation de chaleur et consommation d'énergie* »). Par conséquent, à partir 2005, on a arrêté de chercher à simplement augmenter la fréquence des CPU.

Ainsi, la consommation du CPU est liée à sa fréquence, mais également à sa tension d'alimentation. Les deux sont fortement liées puisque qu'en réussissant à diminuer l'une, on diminue l'autre et on diminue donc la consommation énergétique. « *De plus, la diminution de tension a un effet plus marqué que la diminution de la fréquence. Pour réduire la tension et la fréquence, les ingénieurs ont inventé diverses techniques assez intéressantes, qui permettent d'adapter la tension et la fréquence en fonction des besoins.* » Nous pouvons par exemple citer : (i) la distribution de fréquences et tensions d'alimentations multiples ; (ii) les *Dynamic Voltage* et *Frequency Scaling* (i.e., la variation de la fréquence et de la tension en fonction de ce qui est envoyé au CPU) ; (iii) le *Power Gating* et *Clock Gating* (i.e., la coupure des circuits dits "inutilisés") ; (iv) l'évolution gardée, ressemblant aux Gating (i.e. les circuits sont dits "utiles" sous certaines conditions et donc coupés en fonction).

Pour donner quelques chiffres sur la consommation énergétique de ces différents types d'architecture informatique, nous nous sommes basées sur la littérature. En effet, la consommation énergétique d'une architecture Von Neumann est de l'ordre de 1 nanojoule par opération, celle d'une architecture Harvard est estimée à 0,8 nanojoules (soit 800 picojoules) par opération, alors qu'elle n'est que de 0,01 nanojoule dans les architectures neuromorphiques (Horowitz [2014], Hennessy and Patterson David A. [2011] et Roy et al. [2019]). On peut résumer ces résultats dans le tableau suivant :

11. Cela peut s'expliquer par le fait que l'IPC n'est pas une mesure normalisée (qui donc varie beaucoup d'un système à l'autre).

Architecture	Von Neumann	Harvard	Neuromorphique
Consommation (mJ)	1000	800	10

TABLEAU 1 – Comparaison des valeurs de consommation énergétique en millijoules de différentes architectures informatiques pour un million d’opérations (multiplication de la consommation énergétique typique par opération et du nombre d’opérations).

1.1.2 Les langages de programmation

Les langages de programmation jouent également un rôle indirect mais crucial dans la consommation énergétique des systèmes informatiques. Certains langages sont plus efficaces que d’autres en termes de gestion des ressources et de consommation d’énergie. L’optimisation du code et la sélection du langage approprié contribuent ainsi à réduire la consommation énergétique des applications informatiques.

On distingue 3 types de langages :

- **Les langages compilés** : le code source est traduit en langage machine par un compilateur. L’exécution se fait directement à partir de cette traduction en langage machine, une fois la compilation effectuée (e.g., C, C++, Rust, Fortran, Pascal, Ada).

- **Les langages interprétés** : le code source est interprété ligne par ligne par un logiciel appelé interpréteur. L’exécution se fait en temps réel, directement à partir du code source (e.g., Python, Perl, Ruby, Lua, JavaScript, PHP).

- **Les langages utilisant des machines virtuelles** : le code source est d’abord compilé en bytecode, un code intermédiaire, qui est ensuite exécuté par une machine virtuelle. Cette approche combine des aspects de compilation et d’interprétation (e.g., Java, C#, Scala, Erlang, Dart).

On distingue également différents paradigmes répertoriés ci-dessous.

Les différents langages de programmation présentent des variations en termes de temps de développement, de temps d’exécution, de phase d’initialisation, de phase de recherche et de consommation de mémoire. Selon Prechelt [2000], il est souvent plus rapide de concevoir et d’écrire un programme en Perl, Python, Rexx ou Tcl que de le faire en C, C++ ou Java. Créer un programme dans l’un de ces langages de script peut prendre moitié moins de temps par rapport à l’utilisation des langages compilés, tels que C ou C++. Le code résultant est également beaucoup plus court. Cependant, cette rapidité et cette simplicité ont un coût : *“les programmes en scripts consomment environ deux fois plus de mémoire que ceux en C ou C++”*. Cela signifie que même si les langages de script accélèrent le développement initial et produisent du code plus compact, ils peuvent entraîner une augmentation significative de la consommation de ressources système lors de leur utilisation.

Paradigme	Fonctionnement	Exemples
Impératif	Concentration sur les instructions	C
	Description explicite des étapes	Pascal
	Exécution séquentielle	Fortran
Orienté objet	Concepts d'objet et de classe	Java
	Définition états et comportements	C++
	Modélisation du monde réel	Python
	Réutilisation du code	
Fonctionnel	Fonctions mathématiques	Haskell
	Prévisibles et faciles à tester	OCaml
	Sans états mutables et effets de bord	Lisp
Logique	Logique formelle	Prolog
	Résolution de contraintes	
	Déclaration de faits et de règles	
Script	Automatisation des tâches	JavaScript
	Codes petits et moins structurés	PHP
	Développement rapide	Python

TABLEAU 2 – Paradigmes de programmation, fonctionnement et exemples de langages.

Dans le contexte de ce rapport, la différence qui nous intéresse est l'efficacité énergétique. Dans une étude, Pereira et al. [2017] rappellent l'importance de lever *"une idée fausse, très répandue lors de l'analyse de la consommation d'énergie dans les logiciels, [qui] est qu'elle se comportera de la même manière que le temps d'exécution"*. En effet, l'équation de l'énergie, $\text{Énergie}(J) = \text{Puissance}(W) \times \text{Temps}(s)$ montre que réduire le temps diminue l'énergie consommée. Cependant, la variable Puissance, qui ne peut pas être considérée comme constante, influence également l'énergie consommée. Ainsi, la réduction du temps d'exécution d'un programme n'entraînera pas la même réduction d'énergie.

On remarque que les langages compilés, qui sont les plus rapides, ont aussi tendance à être les plus efficaces. Les langages les plus efficaces énergétiquement sont : C, Rust, C++, Ada 1.70 et Java, et les moins efficaces sont : Lua, Jruby, Ruby, Python et Perl. Ainsi, sur trois algorithmes testés, que sont les binary-trees, fannkuch-redux, et fasta (qui sont respectivement des algorithmes de manipulation de structures de données, de permutations et de génération de séquences), *"les langages compilés consomment 120J pour exécuter les solutions, tandis que les langages de machine virtuelle et interprétés consomment respectivement 576J et 2365J"*.

Nous avons effectué des tests sur un tri par sélection et un tri par tas, en comparant les langages C et Python. Les résultats (cf. Tableau 3) confirment les tendances observées par R. Pereira *et al.* et montrent que C est plus rapide et consomme moins d'énergie que Python. En effet, pour 1 millions de références à trier, C a consommé 16 547,21 J contre une estimation de 70 000 J pour Python, le tout en étant près de 7 fois plus rapide.

		C	Python	Complexité
Tri par sélection	Temps	1091,76	environ 7384	$O(n^2)$
	Joules	16 547,21 5	environ 70 000	
Tri par tas	Temps	0,74	6,5	$O(n \log(n))$
	Joules	11,36	66,26	

TABLEAU 3 – Utilisation de l’outil RAPL pour comparer temps et de énergie consommée par un code de tri par tas et un code de tri par sélection (sur un million de référence) implémentés en C et en Python.

La majorité de l’énergie utilisée est consommée par le CPU, avec une part pouvant varier de 80% à 95% selon les langages. La part de la consommation d’énergie de la mémoire varie quant à elle beaucoup et il est difficile d’établir une corrélation cohérente entre utilisation de la mémoire et énergie lorsqu’on parle de langages. Ainsi, s’il est possible de choisir le meilleur langage lorsqu’on ne considère que le temps et l’énergie, il est plus compliqué de choisir un seul langage dominant quand on inclut la mémoire. Par exemple, C, Pascal, et Go sont équivalents pour le temps et la mémoire, mais C est le meilleur pour l’énergie et le temps. Pour l’énergie et la mémoire, C et Pascal sont équivalents. Lorsqu’on souhaite considérer ces trois caractéristiques en même temps, il faut que le développeur intervienne pour décider quel aspect est le plus important pour son scénario spécifique.







Jun 2024	Jun 2023	Change	Programming Language	Ratings	Change
1	1		 Python	15.39%	+2.93%
2	3	▲	 C++	10.03%	-1.33%
3	2	▼	 C	9.23%	-3.14%
4	4		 Java	8.40%	-2.88%
5	5		 C#	6.65%	-0.06%
6	7	▲	 JavaScript	3.32%	+0.51%
7	14	▲	 Go	1.93%	+0.93%
8	9	▲	 SQL	1.75%	+0.28%
9	6	▼	 Visual Basic	1.66%	-1.67%
10	15	▲	 Fortran	1.53%	+0.53%
11	11		 Delphi/Object Pascal	1.52%	+0.27%
12	19	▲	 Swift	1.27%	+0.33%
13	10	▼	 Assembly language	1.26%	-0.03%
14	12	▼	 MATLAB	1.26%	+0.14%
15	8	▼	 PHP	1.22%	-0.52%
16	13	▼	 Scratch	1.17%	+0.15%
17	20	▲	 Rust	1.17%	+0.26%
18	18		 Ruby	1.11%	+0.17%
19	29	▲	 Kotlin	1.01%	+0.50%
20	22	▲	 COBOL	0.96%	+0.22%

FIGURE 6 – Classement des langages de programmation - juin 2024. Source : TIOBE Softw. BV.

Aujourd'hui, selon l'index TIOBE¹², les langages les plus utilisés sont Python, C++, C, Java, C# et Javascript (cf. Figure 6). Le choix d'un langage de programmation se fait selon les caractéristiques mentionnées plus haut mais aussi en fonction de l'accessibilité du langage, son adaptabilité ou encore des exigences spécifiques du projet.

Nous nous sommes également intéressées à leur compatibilité avec le ML. De nombreuses bibliothèques de ML sont disponibles en Python et offrent une gamme étendue d'outils pour développer des modèles et des algorithmes d'apprentissage automatique. Certaines des bibliothèques les plus populaires incluent TensorFlow, PyTorch, Scikit-learn, Keras et XGBoost. TensorFlow et PyTorch sont des bibliothèques de DL, offrant une flexibilité et des performances exceptionnelles pour la création de réseaux de neurones profonds. Scikit-learn est une bibliothèque polyvalente d'apprentissage automatique offrant une variété d'algorithmes pour la classification, la régression et le clustering. Keras est une interface haut niveau permettant de créer rapidement des réseaux de neurones, souvent utilisée en conjonction avec TensorFlow. XGBoost est une bibliothèque d'implémentation de gradient boosting largement utilisée pour la modélisation prédictive. D'autres langages comme R, Java, C++, Julia, Scala, MATLAB et Go sont également utilisés pour le ML, chacun ayant ses propres avantages en fonction des besoins spécifiques du projet, mais Python reste le choix dominant dans la communauté Data-Science en raison de sa simplicité d'utilisation et de ses puissantes bibliothèques.

1.2 Consommation énergétique de l'intelligence artificielle

Le Machine Learning (ML), qui a révolutionné divers secteurs grâce à ses capacités de prédiction et d'analyse avancées, est également reconnu pour sa consommation énergétique substantielle. Les chercheurs en ML se concentrent pourtant souvent sur la production de modèles précis sans prendre en compte la consommation d'énergie associée.

Les algorithmes de classification, les réseaux de neurones, et d'autres modèles de ML, nécessitent des ressources informatiques intensives, ce qui se traduit par une empreinte carbone notable. Cette section examine la consommation énergétique spécifique au ML et au DL. Nous analyserons également les coûts associés, en utilisant des exemples concrets comme les assistants vocaux et ChatGPT, et comparerons les différentes structures de réseaux de neurones pour évaluer leur efficacité énergétique avant d'effectuer des tests dans la deuxième partie de ce rapport.

12. Le classement du TIOBE Index est déterminé en agrégeant divers facteurs, tels que le nombre de développeurs, de cours de formation et de projets. Les données sont collectées à partir de sources en ligne publiques (moteurs de recherche, forums de développeurs, dépôts de code source, sites d'emploi).

1.2.1 Cas du Machine Learning

a. Définitions

Pour commencer, voici deux tableaux reprenant des informations simples sur l'apprentissage supervisé et non supervisé que nous évoquerons dans ce rapport.

Définition	L'algorithme développe un modèle mathématique à partir d'un ensemble de données contenant toutes les entrées et sorties. Les exemples d'entrée et de sortie souhaités sont prédéfinis.
Principe	L'algorithme reçoit un ensemble d'entrées et les résultats correspondants. Il compare ses résultats réels aux résultats attendus pour ajuster son modèle et minimiser l'erreur.
Problèmes	Résolution de problèmes de régression (valeur continue) Résolution de problèmes de classification (valeur discrète)
Algorithmes	Régression linéaire ou logistique Machine à vecteurs de support (SVM) Arbre de décision
Applications	Prédiction du prix d'un appartement Reconnaissance d'images Détection de spams dans les emails

TABLEAU 4 – Apprentissage supervisé (supervised learning).

Définition	L'algorithme développe un modèle uniquement à partir des entrées. Aucune sortie étiquetée n'est fournie, et les données non étiquetées sont utilisées pour détecter des structures ou des modèles.
Principe	L'algorithme analyse les données pour identifier des regroupements, des anomalies ou des structures cachées sans comparaison avec des valeurs réelles.
Problèmes	Regroupement Réduction de dimensionnalité Détection d'anomalies
Algorithmes	Apprentissage de modèles de variables latentes Clustering
Applications	Identification de plusieurs types de bactéries Segmentation de marché en merchandising Détection de fraudes bancaires

TABLEAU 5 – Apprentissage non supervisé (unsupervised learning).

Bien que nous n'ayons pas fait nos recherches spécifiquement sur ce sujet et que nous n'ayons donc pas de résultats originaux en lien, il existe un troisième type de ML : l'apprentissage par renforcement. Dans ce cas, la machine apprend en interagissant avec un environnement, en recevant des récompenses ou des punitions en fonction des actions qu'elle entreprend. Il traite notamment des problèmes d'optimisation de politiques, de prise de décision séquentielle et d'apprentissage de comportements complexes dans les voitures autonomes, la prise de décision ou encore les jeux.

Trois phases importantes sont à distinguer dans l'élaboration et l'utilisation des modèles de ML :

La **phase de prétraitement** des données consiste à préparer et nettoyer les données pour qu'elles soient exploitables par les machines.

La **phase d'apprentissage**, également connue sous le nom de phase de formation ou d'entraînement, est l'étape du processus où le modèle est exposé à des données d'entraînement afin d'apprendre des schémas et des motifs utiles. Pendant cette phase, le modèle ajuste ses paramètres internes pour minimiser une fonction de perte, ce qui lui permet de s'adapter aux données et d'acquérir des capacités prédictives.

La **phase de prédiction ou d'inférence** pendant laquelle le modèle est utilisé pour faire des prédictions ou des classifications sur de nouvelles données en fonction de ce qu'il a appris pendant les phases d'entraînement. Cette phase est souvent réalisée de manière itérative et en temps réel dans des applications pratiques telles que la reconnaissance d'images, la traduction automatique, ou les recommandations personnalisées.

Illustration avec le cas du NLP (*Natural Language Processing*)

Le NLP, ou *traitement automatique du langage*, est utilisé par les assistants vocaux après la reconnaissance vocale ou encore par la traduction automatique.

Lors du prétraitement, le modèle des *sacs de mots* compte les mots du texte en créant une matrice d'occurrence, sans tenir compte de la syntaxe ou de la sémantique, afin de constituer un classificateur des caractéristiques, bien que cette approche ne permette d'analyse contextuelle ou sémantique. La *tokenisation* segmente le texte en phrases ou en mots, appelés « tokens », et élimine les caractères comme les ponctuations. La *stemming* coupe les préfixes et suffixes des mots pour les réduire à leur forme de base. La *lemmatisation* réduit les mots à leur forme de base en regroupant les différentes formes d'un même mot, nécessitant des dictionnaires détaillés pour relier les mots à leurs radicaux tout en prenant en compte le contexte pour identifier leur signification. La *suppression des stop words*, ou mots vides, consiste à enlever les articles, pronoms et prépositions pour libérer de l'espace et accélérer le traitement des données. La *modélisation des sujets* découvre les structures cachées dans les contenus en présumant que chaque document se compose d'un mélange de sujets.¹³

13. <https://intelligence-artificielle.com/nlp-guide-complet/>

Après le prétraitement des données, l'étape suivante du NLP est le développement d'un algorithme pour les interpréter, c'est la phase d'apprentissage. Les trois types d'approches couramment utilisées sont basées sur les règles, le ML et le DL. La méthode basée sur les règles utilise des règles linguistiques spécifiques à un domaine pour résoudre des problèmes simples, comme classer les mails indésirables en filtrant des termes spécifiques. La méthode basée sur le ML entraîne un système avec des données pour qu'il apprenne à effectuer des tâches complexes, utilisant des méthodes statistiques pour comprendre le langage. Enfin, la méthode basée sur le DL utilise plusieurs couches de réseaux neuronaux, permettant d'effectuer des tâches difficiles comme la traduction sans nécessiter de prétraitement complexe, grâce à ses extracteurs automatiques des caractéristiques.

Enfin, la phase d'inférence consiste à l'utilisation du modèle (Barthès [2004]).

b. Consommation énergétique

La consommation énergétique des algorithmes peut varier considérablement selon leur complexité et selon la taille des ensembles de données utilisés. Des algorithmes simples comme les *k-plus proches voisins* (k-NN) ou les *machines à vecteurs de support* (SVM) consomment généralement moins d'énergie comparativement aux modèles plus complexes de type forêts aléatoires. De plus, les différentes phases (d'apprentissage et de prédiction) du ML ont par ailleurs des consommations différentes.

Coûts énergétiques pendant l'entraînement :

Une manière de calculer la quantité de CO₂eq (C) émis pendant l'entraînement du modèle est proposée par Luccioni et al. [2022]. Elle consiste à décomposer C en trois facteurs pertinents : la consommation d'énergie du matériel utilisé (P), le temps d'entraînement (T) et l'intensité carbone de la grille énergétique (I) ; ou de manière équivalente, l'énergie consommée (E) et l'intensité carbone :

$$C = P \times T \times I = E \times I$$

L'**intensité carbone**, ou facteur d'émission de carbone, mesure la quantité de dioxyde de carbone (CO₂) émise pour produire une unité d'énergie, généralement exprimée en grammes de CO₂ par kilowattheure (gCO₂/kWh) d'électricité générée. Des sources publiques telles que l'*Agence Internationale de l'Energie* et l'*Administration de l'Information sur l'Energie* permettent d'obtenir des informations sur l'intensité carbone des réseaux énergétiques dans différentes régions du monde. Pour les modèles entraînés sur des plateformes de cloud computing commerciales telles que Google Cloud ou Amazon Web Services (AWS), il est possible d'utiliser les informations fournies par ces entreprises pour estimer les facteurs d'émission. La **consommation d'énergie du matériel** est calculée via la puissance thermique de conception qui indique l'énergie dont il a besoin sous la charge théorique maximale. Enfin, le **temps d'entraînement** est calculé comme le nombre total d'heures de matériel, c'est-à-dire que 16 GPU utilisés pendant 24h et 8 GPU utilisés pendant 48h auront le même temps d'entraînement de 384 heures-GPU.

Coûts énergétiques et empreinte carbone pendant l'inférence :

Les calculs se concentrent souvent sur la phase d'entraînement et moins sur la phase d'inférence. Or, en 2022, 1/3 de l'empreinte carbone liée au ML de l'entreprise Meta en interne était dû à l'inférence des modèles, tandis que chez Google, 60% de la consommation d'énergie était dû à l'inférence des modèles contre 40% à l'apprentissage de ces derniers. En effet, *"bien que l'inférence sur un seul exemple nécessite beaucoup moins de calculs que l'entraînement du même modèle, l'inférence se produit beaucoup plus fréquemment que l'entraînement du modèle"* (Luccioni et al. [2024]).

	Classification	Génération	Tâches multimo- dales
Exemples de tâches	Classification d'images et de texte	Génération de texte et résumé	Légende d'images et génération d'images
Consommation pour 1000 inférences (kWh)	entre 0.002 et 0.007	environ 0.05	entre 0.06 et 2.9

TABLEAU 6 – Résultats de la consommation énergétique des algorithmes dans l'étude de Luccioni et al. [2024].

En utilisant des outils de mesure que nous détaillerons dans la partie 2.), la consommation énergétique et l'empreinte carbone de différents modèles peuvent être calculés. Ainsi, A. Luccioni *et al.* ont étudié des modèles répartis sur 10 tâches couvrant des applications en langage naturel et en vision par ordinateur.

En comparaison, charger un smartphone moyen nécessite 0.012 kWh d'énergie. Ainsi, le modèle de génération de texte le plus efficace consomme **autant d'énergie que 16% d'une charge complète de smartphone** pour 1000 inférences, tandis que le modèle de génération d'image le moins efficace consomme autant d'énergie que 950 charges de smartphone. Cette différence peut être attribuée à la diversité des tâches et à la complexité des modèles utilisés ou encore à la taille du modèle. Les modèles de génération d'images sont les plus énergivores et peuvent émettre **jusqu'à 1 594 grammes de CO2 pour 1 000 inférences**, soit l'équivalent de 4.1 miles parcourus par un véhicule à essence moyen. En revanche, les modèles de génération de texte sont beaucoup plus économes en énergie, émettant jusqu'à l'équivalent de 0.0006 miles de CO2 pour 1 000 inférences, soit 6 833 fois moins.

Empreinte carbone totale :

Une formule proposée par Patterson et al. [2022] permet de prendre en compte les coûts d'entraînement, les coûts d'inférences et l'énergie consommée par les data centers.

$$\text{Footprint} = (\text{electrical energy}_{\text{train}} + \text{queries} \times \text{electrical energy}_{\text{inference}}) \times \text{CO2}_{\text{datacenter}} \setminus \text{kWh}$$

Les calculs de coûts d'énergie et d'empreinte carbone prennent souvent donc en compte à la fois les composantes du hardware et celles du software.

1.2.2 Cas du Deep Learning

a. Définitions

Les réseaux de neurones, sous catégorie de ML, en particulier les *réseaux de neurones profonds* (Deep Neural Networks ou DNN), sont connus pour leur capacité à traiter des tâches complexes comme la reconnaissance d'image et la génération de langage naturel. Cependant, cette puissance de traitement se traduit par une consommation énergétique élevée.

Pour rappel, l'objectif de l'apprentissage automatique est de déterminer la meilleure solution au problème posé en manipulant les paramètres d'un modèle sur la base de nombreuses données d'entraînement (c'est-à-dire trouver le modèle qui s'ajuste le mieux aux données fournies à la machine). Les réseaux de neurones se caractérisent donc par des paramètres ou poids utilisés pour transformer les données d'entrée en caractéristiques (cf. Figure 7).

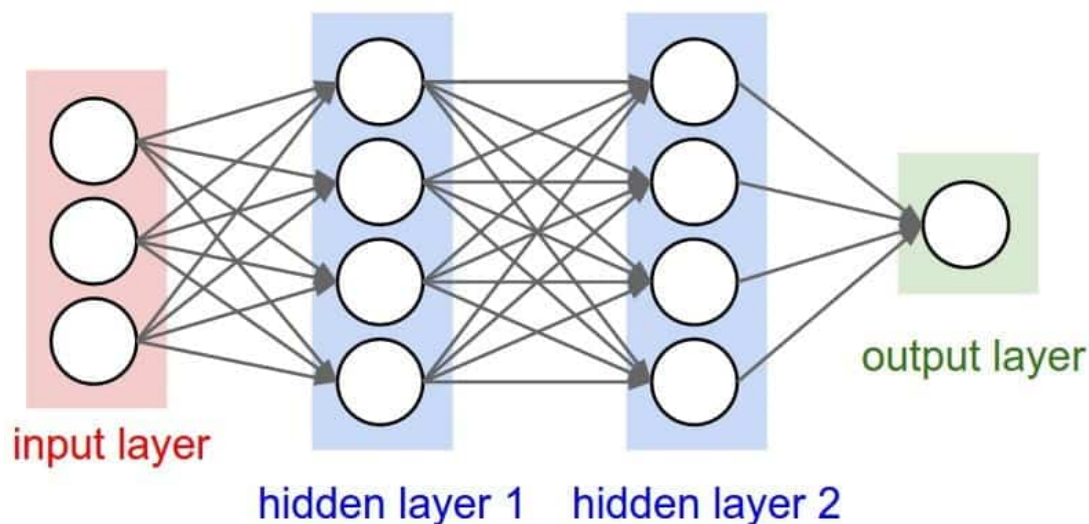


FIGURE 7 – Représentation générale d'un réseau de neurones artificiels. Source : Lebig-data.

Les DNN peuvent être décomposés en deux phases de calculs distinctes : (i) *la phase d'apprentissage* et (ii) *la phase d'inférence*. Lors de cette première phase, le DNN est créé. On précise son nombre de couches, sa taille et le type de couche. Le réseau « apprend » ensuite les paramètres de poids. Lors de la deuxième phase, ces paramètres de poids deviennent fixes et on teste le modèle sur de nouvelles données d'entrées, afin d'extraire ce qu'on appelle des "*caractéristiques de données d'entrée*" (García-Martín et al. [2019] et Yang et al. [2017]).

Parmi les grands types de modèles d'apprentissage automatique, on trouve : (i) les *perceptrons multicouches* ou **MLP** ; (ii) les *réseaux de neurones convolutionnels* ou **CNN** ; (iii) les *réseaux de neurones récurrents* ou **RNN** ; (iv) les *réseaux antagonistes génératif* ou **GAN** ; (v) les **transformers** (par exemple utilisés dans ChatGPT). Comme dans la section précédente, nous pouvons résumer leurs principes et caractéristiques dans des tableaux¹⁴ :

Définition	Ce modèle est composé de plusieurs couches de neurones, dont nécessairement une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie.
Principe	L'algorithme permet l'apprentissage de relations complexes entre données d'entrée et données de sortie, grâce à des fonctions d'activation non linéaire. Chaque neurone d'une couche est connecté à chaque neurone de la couche suivante. Les algorithmes mathématiques entrant en jeu sont les suivants : forward propagation , backpropagation et algorithme de minimisation de la fonction coût .
Problèmes	Surapprentissage (modèle trop complexe pour le peu de données disponibles) Réduction de la taille des gradients (ralentissement de l'apprentissage) Gourmands en ressources (grands besoins en calculs si les tâches sont complexes)
Applications	Classification d'images Prédiction des prix de l'immobilier Détection de fraudes

TABLEAU 7 – Les perceptrons multicouches ou **MLP** (Explications complètes sur : <https://neuroconnection.eu/quand-utiliser-les-reseaux-de-neurones-mlp-cnn-et-rnn/>).

14. Ces modèles sont détaillés ici car ils seront implémentés et testés plus tard dans le rapport.

Définition	Ce modèle (inspiré du cortex visuel animal) est conçu pour reconnaître des données disposées en grille, comme les images (bidimensionnelles, champ ou matrice). Il constitue la référence pour les prédictions pour des données d'images en entrées.
Principe	L'algorithme fonctionne en appliquant des opérations sur les données d'entrée. Il y en a 3 types : la convolution , donnant une « carte d'activation » des caractéristiques importantes de l'image ; le pooling (en général maxpooling), qui réduit la dimensionnalité ; et la fonction d'activation de type ReLU.
Problèmes	Surajustement (ou overfitting) si les données en entrée sont limitées Sensible aux déformations et changements d'échelles des images Gourmand en ressources (besoin de données annotées pour entraînement)
Algorithmes	LeNet (reconnaissance de caractères manuscrits dans chèques) GoogLeNet (Inception)
Applications	Reconnaissance d'images, vocale Traduction automatique Segmentation sémantique

TABLEAU 8 – Les réseaux de neurones convolutionnels ou **CNN** (Explications complètes sur : <https://fr.blog.businessdecision.com/tutoriel-deep-learning-le-reseau-neuronal-convolutif-cnn/>).

Définition	Ce modèle est conçu pour traiter des données séquentielles (texte, audio, etc.). Grâce à ses connexions récurrentes entre les neurones, il conserve une sorte " <i>de mémoire interne</i> ".
Principe	L'algorithme fonctionne en créant des boucles dans l'architecture du réseau. Ce dernier prend en compte à la fois les données en entrée (données actuelles) et la mémoire de l'étape précédente (état caché). L'algorithme fait alors des prédictions sur la base des motifs (puisque'il garde en mémoire les séquences précédentes).
Problèmes	Disparition du gradient (si devient trop petit) Explosion du gradient (si devient trop grand)
Algorithmes	LSTM (Réseaux de mémoire à court terme) GRU (Unité récurrente fermée)
Applications	Reconnaissance de parole Traduction automatique Génération de texte

TABLEAU 9 – Les réseaux de neurones récurrents ou **RNN** (Explications complètes sur : <https://datavalue-consulting.com/deep-learning-reseaux-neurones-recurrents-rnn/>).

Définition	Ce modèle est conçu pour créer des données synthétiques telles que des images réalistes. Il est composé de deux réseaux de neurones en compétition.
Principe	L'algorithme fonctionne grâce à ces deux réseaux : : <i>le générateur</i> (créé des données réalistes ou non) et <i>le discriminateur</i> (discrimine les données en fonction de leur caractère réaliste par rapport aux données d'entrées). Au fur et à mesure de l'entraînement, ils deviennent tous deux plus compétents.
Problèmes	Convergence lente (le temps de convergence des 2 réseaux est très long) Instabilité de l'entraînement (équilibre non-stable entre les 2 réseaux) Mode collapse (nombre limité de résultats réalistes)
Algorithmes	VanillaGAN (configuration de base) DCGAN (intègre des CNN dans le traitement d'images)
Applications	Génération d'image réaliste, de texte Restauration d'images endommagées Création de vidéos synthétiques

TABLEAU 10 – Les réseaux antagonistes génératifs ou **GAN** (Explications complètes sur : <https://khayyam.developpez.com/articles/intelligence-artificielle/tensorflow-gan/>).

Définition	C'est un type d'apprentissage profond, qui n'est apparu que récemment ("Attention is all you need", 2017), conçu pour traiter le langage naturel (maintenant utilisé pour la vision par ordinateur, le traitement de séquences, etc.)
Principe	L'algorithme ne dépend pas d'une architecture récurrente ou convolutive (comme RNN ou CNN) mais d'une " <i>architecture d'attention qui prend en compte toutes les positions dans la séquence d'entrée en même temps.</i> " Il se compose en général d'un encodeur et/ou d'un décodeur.
Problèmes	Coût computationnel élevé Difficultés à capturer les dépendances à long terme
Algorithmes	BERT (<i>Bidirectional Encoder Representations from Transformers</i>) GPT (<i>Generative Pre-trained Transformer</i>)
Applications	Traduction automatique Génération de texte, de code, d'images Reconnaissance de la parole

TABLEAU 11 – Les **transformers** (utilisés par ex. dans ChatGPT) (Explications complètes sur : <https://france.devoteam.com/paroles-dexperts/lstm-transformers-gpt-bert-guide-des-principales-techniques-en-nlp/>).

b. Consommation énergétique

La formation de réseaux de neurones nécessite généralement l'utilisation de GPU (unités de traitement graphique) et/ou de TPU (unités de traitement tensoriel), qui sont des *"accélérateurs matériels spécialisés conçus pour traiter efficacement des tâches de calcul spécifique"*¹⁵, gourmands en énergie. En outre, les coûts énergétiques augmentent de manière exponentielle avec la profondeur et la complexité des réseaux. Ainsi, les CNN, GAN et Transformers ont tendance à être plus coûteux que les MLP et les RNN puisque leurs architectures sont plus complexes et qu'elles ont un plus grand nombre de paramètres. Cependant, le coût d'un MLP dépendra du nombre de couches cachées et de neurones par couche et celui d'un RNN, de la longueur de la séquence à traiter (plus la séquence est longue, plus elle nécessite d'itérations). Lors de la phase d'entraînement, ces modèles reposent principalement sur des systèmes de bureau ou des serveurs (y compris des GPU, CPU ou FPGA haut de gamme), ce qui consomme beaucoup (Hu et al. [2022]). Alors que la phase d'inférence *"est généralement effectuée sur des systèmes embarqués bas de gamme, par exemple des téléphones intelligents, des appareils portables et autres"*. Elle est en général plus rapide et utilise moins de puissance de calcul. Dans certains cas, la phase d'inférence peut être plus coûteuse : c'est le cas des modèles GAN et Transformers (García-Martín et al. [2019] et Yang et al. [2017]). Un outil comme ChatGPT nécessite non seulement *"une utilisation de vastes ensembles de données et de ressources de calcul intensives"* mais consomme aussi lors des prédictions. En effet, *"la formation d'un seul LLM comme le GPT-3 a nécessité environ 10 GWh, soit environ 6 000 fois l'énergie qu'un citoyen européen consomme par an"*.

★ ★ ★

15. Source : <https://exittechnologies.com/fr/blog/it-tips/tpu-vs-gpu-unraveling-the-power-struggle/>

2 - Techniques de Green Computing

Il est crucial d'adopter des techniques de **green computing** pour minimiser l'impact environnemental des technologies. Nous allons faire un tour d'horizon de ce qu'il est possible de mettre en place dans cette partie.

Le concept de green computing, ou informatique verte, a émergé dans les années 1990 avec le lancement du programme *Energy Star* par l'Agence de protection de l'environnement (EPA) aux États-Unis. Ce programme a été créé pour promouvoir et reconnaître l'efficacité énergétique au sein des entreprises. Les produits certifiés *Energy Star* doivent respecter des critères spécifiques, incluant des fonctions avancées de gestion de l'énergie, souvent absentes des produits non certifiés.

Le green computing désigne l'ensemble des pratiques et des technologies visant à rendre l'utilisation des ressources informatiques plus durables et respectueuses de l'environnement. Cette approche englobe plusieurs aspects, allant de la conception et de la fabrication de matériels plus économes en énergie, à l'optimisation des logiciels pour minimiser leur consommation énergétique. Les objectifs principaux du green computing sont de réduire les déchets électroniques, diminuer l'empreinte carbone des infrastructures informatiques, et promouvoir une utilisation plus efficiente des ressources énergétiques.

2.1 Pour réduire la consommation d'énergie de l'intelligence artificielle

Nous examinerons d'abord les stratégies appliquées au sein des algorithmes de ML et de DL. Dans cette partie, nous avons effectué des programmes que nous avons testés, et qui se situent pour la plupart en annexe et/ou sur notre GitHub : https://github.com/RosalieJA/TX00_GREEN_AI/tree/main.

2.1.1 Au sein des algorithmes de Machine Learning

Comme vu dans la partie précédente, les algorithmes de ML peuvent générer une consommation d'énergie intensive. Pour réduire cette consommation, plusieurs approches peuvent être adoptées.

a. Utilisation d'outils de mesure

Afin de connaître la consommation énergétique ou l'empreinte carbone d'un code, il peut être intéressant d'utiliser des outils permettant de faire des mesures efficaces.

RAPL

RAPL (*Running Average Power Limit*), est un outil puissant utilisé pour évaluer la consommation énergétique des processeurs et des autres composants matériels dans un système informatique. Il permet de surveiller et de contrôler la consommation électrique en temps réel, offrant ainsi aux développeurs et aux ingénieurs une visibilité précise sur la façon dont le code qu'ils écrivent impacte l'efficacité énergétique du système. En mesurant les données telles que la puissance instantanée, la consommation énergétique totale et d'autres métriques pertinentes, le RAPL permet d'optimiser les performances tout en minimisant la consommation d'énergie (Khan et al. [2018]).

CodeCarbon

*CodeCarbon*¹⁶ est un outil conçu pour quantifier l'impact environnemental des logiciels. Il calcule l'empreinte carbone d'un logiciel en prenant en compte la consommation d'énergie nécessaire à son exécution sur différents types de matériel, ainsi que les émissions de carbone associées à cette consommation d'énergie. De manière générale, comme mentionné lors de certains calculs précédents, pour les données sur l'intensité carbone de l'énergie, CodeCarbon utilise Google pour Google Cloud Platform, mais Amazon et Microsoft ne publient pas de détails spécifiques sur leurs centres de données. Pour les infrastructures privées, l'outil utilise les données de "Our World in Data" ou le mix énergétique national de globalpetrolprices.com, multiplié par l'intensité carbone de la source d'électricité, avec une moyenne mondiale de 475 gCO₂.eq/KWh de l'IEA en cas d'absence de données.

Calculateur de Green Algorithms

D'autres outils sont également disponibles afin de calculer la consommation d'un programme. *Green Algorithms*¹⁷ est un projet visant à promouvoir une science informatique plus respectueuse de l'environnement et qui propose un calculateur en ligne¹⁸ permettant d'estimer l'empreinte carbone des algorithmes.

Le calcul de l'empreinte carbone (C) est effectué de cette manière :

$$C = t \times (n_c \times P_c \times u_c + n_m \times P_m) \times PUE \times CI \times 0.001$$

Où t représente le temps d'exécution (en heures), n_c le nombre de cœurs, n_m la taille de la mémoire disponible (en gigaoctets), u_c est le facteur d'utilisation du cœur (compris entre 0 et 1), P_c est la consommation énergétique d'un cœur de calcul, P_m la consommation énergétique de la mémoire (en watts), PUE est le coefficient d'efficacité du data center et CI l'intensité carbone (en gCO₂e/kWh).

16. <https://codecarbon.io>

17. <https://www.green-algorithms.org>

18. <http://calculator.green-algorithms.org>

b. Choix des algorithmes

En choisissant des algorithmes plus efficaces en termes de calcul ou adaptés à la tâche à effectuer, on peut réduire la consommation d'énergie. Ce choix peut se faire selon différents critères. Nous avons effectué deux comparaisons : l'une entre apprentissage supervisé et non supervisé, l'autre entre ML et modèle mathématique.

Supervisé VS non supervisé

Nous avons implémenté deux algorithmes pour reconnaître des chiffres manuscrits issus de la base de données MNIST sur laquelle nous reviendrons dans la partie 2.1.2 et effectué nos tests avec CodeCarbon.

L'algorithme **KNN est un algorithme d'apprentissage supervisé** qui classe un point de test en fonction de la majorité des k points les plus proches de ce point dans le jeu de données d'entraînement (cf. Figure 8) . La principale source de consommation d'énergie réside dans le calcul des distances entre le point de test et tous les points du jeu de données d'entraînement. Si le jeu de données est volumineux, cela peut entraîner une consommation d'énergie significative, en particulier sur des architectures matérielles où le calcul de distances est coûteux en énergie.

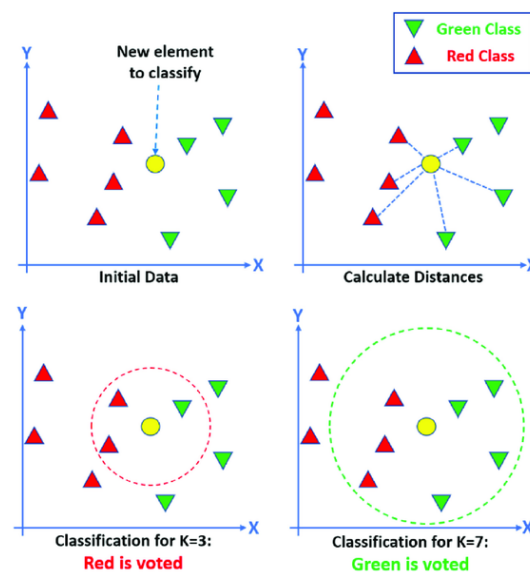


FIGURE 8 – Fonctionnement de l'algorithme KNN. Source : Zidi et al. [2023].

L'algorithme **KMeans est un algorithme d'apprentissage non-supervisé** dont le fonctionnement consiste à partitionner les données en k groupes en divisant des points représentant les données en k groupes (les *clusters*) de façon à minimiser une certaine fonction (cf. Figure 9). La consommation d'énergie est influencée par plusieurs facteurs, y compris par le nombre de clusters k et le nombre d'itérations nécessaires pour atteindre la convergence. Des valeurs élevées de k ou un grand nombre d'itérations peuvent entraîner une consommation d'énergie plus élevée.

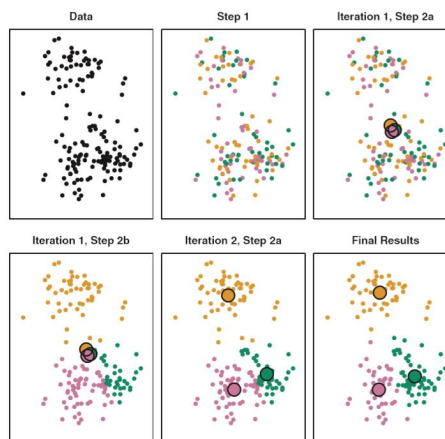


FIGURE 9 – Fonctionnement de l'algorithme KMeans. Source : Della Data.

Nous avons testé l'algorithme KNN avec 3 voisins et KMeans avec 256 clusters (nombres qui permettaient une plus grande précision). Nous avons donc expérimentalement observé que pour effectuer une même tâche, l'algorithme de ML supervisé est moins consommateur tout en étant plus efficace lors de la phase d'inférence. Nos résultats montrent également que l'algorithme de ML supervisé est beaucoup plus rapide lors de sa phase d'entraînement (cf. Tableau 12).

	KNN	KMeans
Emissions (kg CO2)	1.0702234314084056e-08	4.983156900972128e-07
Précision	0.9447	0.9014
Temps (s)	0.10460591316223145	9.707077741622925

TABLEAU 12 – Résultats des émissions des algorithmes KNN et KMeans.

Machine learning VS modèle

Bien que l'IA soit un sujet d'actualité et qu'il peut être tentant d'utiliser des algorithmes de ML, certains modèles "plus basiques" permettent d'effectuer aussi bien les mêmes tâches tout en consommant moins. A titre d'illustration, nous avons implémenté deux algorithmes qui génèrent une image très simple.

La **transformée de Fourier** est une technique utilisée en traitement du signal et en analyse de données pour décomposer une fonction périodique en une somme de fonctions sinusoïdales de différentes fréquences. Elle permet de passer d'un signal dans le domaine temporel à son équivalent dans le domaine fréquentiel. La consommation d'énergie dépend principalement de la taille de l'image et du nombre de fréquences utilisées. Si l'image est grande ou si un grand nombre de fréquences sont utilisées, cela peut entraîner une consommation d'énergie plus élevée en raison des calculs intensifs nécessaires pour effectuer la transformée et son inverse.

L'algorithme **KMeans**, utilisé pour la segmentation d'image, consiste à diviser une image en plusieurs régions ou segments afin de simplifier son analyse ou son traitement ultérieur. L'algorithme KMeans nécessite une initialisation des centroïdes et des itérations pour atteindre la convergence, ce qui peut entraîner une consommation d'énergie plus élevée, surtout pour des images de grande taille ou avec un grand nombre de clusters.

	Transformée de Fourier	KMeans
Emissions (kg CO2)	1.7219968071579936e-09	4.493341171741487e-09
Temps d'exécution (s)	0.0035228729248046875	0.02617812156677246

TABLEAU 13 – Résultats des émissions des algorithmes utilisant Fourier et KMeans.

D'après nos résultats (cf. Figure 13), il apparaît que pour réaliser la même tâche, le modèle exploitant la transformée de Fourier s'avère à la fois plus rapide et moins énergivore que l'algorithme du KMeans. Cela met en lumière l'efficacité de la transformée de Fourier dans ce contexte spécifique, offrant ainsi une solution efficiente du point de vue énergétique pour le traitement d'images.

On peut déjà tirer quelques conclusions d'après les résultats de nos tests. Pour minimiser l'empreinte carbone et améliorer l'efficacité énergétique des algorithmes de Machine Learning, il peut être utile de :

- d'utiliser des outils de mesure adéquats pour évaluer et comprendre la consommation énergétique ;
- de choisir les algorithmes à utiliser en fonction de leur efficacité énergétique, en tenant compte du compromis entre précision et consommation ;
- de considérer des modèles plus simples à chaque fois que cela est possible, car ils peuvent offrir des performances comparables avec une consommation énergétique réduite.

2.1.2 Au sein des algorithmes de Deep Learning

Les algorithmes de DL, en raison de leur complexité et de la taille des modèles, consomment encore plus d'énergie que les algorithmes vus précédemment. Nous avons ici décidé d'implémenter en Python des réseaux de neurones (un par grand type présenté dans la partie 1.2.2.), de calculer et de comparer leur consommation énergétique au moment de l'entraînement et de comparer l'exactitude de leur prédiction.

a. Implémentation de réseaux de neurones artificiels

Pour cela, nous avons utilisé les outils présentés dans la partie précédente (Code Carbon, Green Algorithm). Nous avons entraîné ces réseaux de neurones sur la base

de données MNIST, une base de données de chiffres écrits à la main, très utilisée en Deep Learning (cf. Figure 10) et nous avons ensuite comparé leurs prédictions sur un set d'images MNIST indépendantes (non utilisées pendant l'entraînement). Nous avons aussi utilisé les bibliothèques TensorFlow, utilisées pour développer des réseaux de neurones. Ainsi, nous avons commencé par comparer les réseaux suivants : un MLP, un CNN et un RNN.

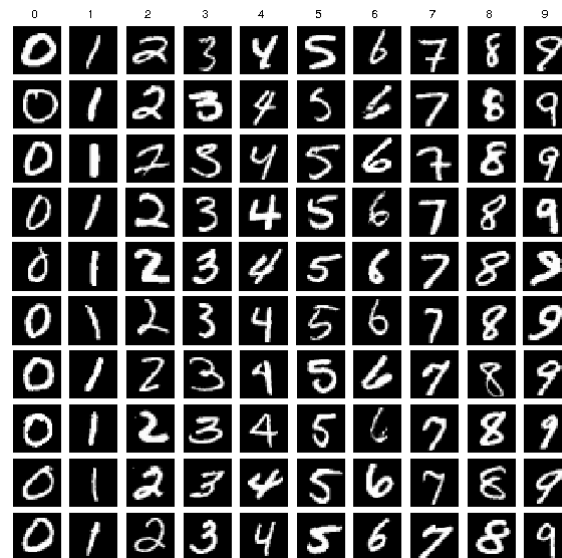


FIGURE 10 – Base de données MNIST. Source : NIST.

Description	Valeurs
Énergie consommée par la RAM	0.000031 kWh
Puissance de la RAM	2.9302897453308105 W
Énergie consommée par tous les CPU	0.000134 kWh
Puissance totale du CPU	12.5 W
Électricité utilisée depuis le début	0.000165 kWh
Émissions	1.4007240778055813e-05 kg CO2
Précision du test	0.9797999858856201
Nombre de bonnes réponses	7 sur 8
Temps	41 secondes

TABEAU 14 – Données de performances et de consommation d'énergie obtenues à l'aide de CodeCarbon, suite à l'entraînement du MLP.

Pour cela, nous avons choisi de créer un perceptron multi-couches avec 10 couches cachées¹⁹. A noter que lorsque l'on joue à ajouter des couches cachées, le MLP consomme plus d'énergie. Cependant, cela ne signifie pas que la prédiction sera nécessairement meilleure (puisqu'il existe un risque de surapprentissage). A l'aide de notre programme,

¹⁹. Pour créer notre MLP, nous nous sommes aidées de ce site et de la documentation officielle de TensorFlow : <https://khayyam.developpez.com/articles/intelligence-artificielle/tensorflow/>

se trouvant en Annexe 3), nous avons obtenu les résultats répertoriés dans le tableau 14 et dans la figure 11.

```

1/1 [=====] - 0s 244ms/step
zero.png => 0
|
1/1 [=====] - 0s 49ms/step
one.png => 1

1/1 [=====] - 0s 52ms/step
two.png => 2

1/1 [=====] - 0s 65ms/step
three.png => 3

1/1 [=====] - 0s 52ms/step
four.png => 4

1/1 [=====] - 0s 50ms/step
five.png => 3

1/1 [=====] - 0s 63ms/step
six.png => 6

1/1 [=====] - 0s 57ms/step
seven.png => 7

```

FIGURE 11 – Résultats de la prédiction du MLP entraîné.

Il est intéressant de noter que CodeCarbon ne pouvait pas accéder au GPU. En effet, nos ordinateurs ne possédaient pas de GPU supplémentaire (de type NVIDIA), ce qui nous a également empêché de tester pyRAPL, outil RAPL pour python²⁰. De plus, le modèle du CPU utilisé ici (et pour tous nos tests) est le suivant : Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz.

Description	Valeurs
Énergie consommée par la RAM	0.000243 kWh
Puissance de la RAM	2.9302897453308105 W
Énergie consommée par tous les CPU	0.001038 kWh
Puissance totale du CPU	12.5 W
Électricité utilisée depuis le début	0.001281 kWh
Émissions	0.00010874017996317586 kg CO2 eq
Précision du test	0.9869999885559082
Nombre de bonnes réponses	8 sur 8
Temps	302 secondes

TABLEAU 15 – Données de performances et de consommation d'énergie obtenues à l'aide de CodeCarbon, suite à l'entraînement du GRU.

Nous avons ensuite implémenté un réseau neuronal récurrent (situé en Annexe 3). Une fois de plus, le nombre de couches cachées joue un rôle majeur quand à la consommation énergétique (ici, il y en a 2 de type GRU). La taille des couches est aussi importante,

20. Nous souhaitions au départ comparer les résultats de CodeCarbon et RAPL.

puisque plus cette dernière est grande, plus grande sera la consommation (ici, il y a 128 neurones par couche). Enfin, nous sommes dans le cas d'un **GRU** (*Gated Recurrent Units*), un type spécifique de RNN, dont l'architecture est plus sophistiquée qu'un RNN classique (gérant mieux les problèmes d'explosion et de disparition de gradient, cf. Tableau 9) mais qui par conséquent consomme plus. A l'aide de notre programme, nous avons obtenu les résultats répertoriés dans le tableau 15 et dans la figure 12.

```

1/1 [=====] - 1s 1s/step
zero.png => 0

1/1 [=====] - 0s 45ms/step
one.png => 1

1/1 [=====] - 0s 44ms/step
two.png => 2

1/1 [=====] - 0s 42ms/step
three.png => 3

1/1 [=====] - 0s 42ms/step
four.png => 4

1/1 [=====] - 0s 55ms/step
five.png => 5

1/1 [=====] - 0s 41ms/step
six.png => 6

1/1 [=====] - 0s 50ms/step
seven.png => 7

```

FIGURE 12 – Résultats de la prédiction du RNN entraîné.

Nous avons enfin implémenté un réseau de neurones convolutifs avec 7 couches cachées (situé en Annexe 3). En plus du nombre de couches cachées, son architecture étant complexe, le CNN devrait consommer plus que les deux réseaux précédents. A l'aide de notre programme, nous avons obtenu les résultats répertoriés dans le tableau 16 et dans la figure 13.

Description	Valeurs
Énergie consommée par la RAM	0.000084 kWh
Puissance de la RAM	2.9302897453308105 W
Énergie consommée par tous les CPU	0.000359 kWh
Puissance totale du CPU	12.5 W
Électricité utilisée depuis le début	0.000444 kWh
Émissions	3.7664445672275096e-05 kg CO2
Précision du test	0.9904999732971191
Nombre de bonnes réponses	7 sur 8
Temps	205 secondes

TABLEAU 16 – Données de performances et de consommation d'énergie obtenues à l'aide de CodeCarbon, suite à l'entraînement du CNN.

```

1/1 [=====] - 0s 158ms/step
zero.png => 0

1/1 [=====] - 0s 25ms/step
one.png => 1

1/1 [=====] - 0s 31ms/step
two.png => 2

1/1 [=====] - 0s 36ms/step
three.png => 3

1/1 [=====] - 0s 29ms/step
four.png => 4

1/1 [=====] - 0s 27ms/step
five.png => 3

1/1 [=====] - 0s 25ms/step
six.png => 6

1/1 [=====] - 0s 22ms/step
seven.png => 7

```

FIGURE 13 – Résultats de la prédiction du CNN entraîné.

Si l'on compare ensuite les 3 valeurs correspondant aux émissions en kg CO₂ eq, on remarque alors que le MLP est celui qui consomme le moins, mais qu'il a une moins bonne prédiction. Si on s'intéresse plus précisément aux RNN et CNN, on remarque que l'empreinte carbone est bien meilleure pour le CNN alors qu'il a la meilleure précision du test. Cela peut s'expliquer par le fait que l'objectif principal des CNN est de traiter des images, ce qui réduit énormément le nombre de paramètres à entraîner. Cela peut se traduire par une utilisation plus efficace des ressources matérielles. Avec un autre exemple, les résultats auraient peut-être été différents.

Une première conclusion évidente que l'on peut tirer de cela est que :

- il faut faire parfois des compromis entre précision et consommation énergétique. Augmenter le nombre de couches cachées, et donc la consommation, ne se traduira pas forcément une majoration de la performance ;
- il faut utiliser le type de réseau adapté même s'il est plus complexe en terme d'architecture, car cela peut faire gagner en consommation énergétique et en précision (voir CNN vs RNN) ;
- il faut utiliser des outils comme RAPL ou CodeCarbon lors de la création de son algorithme afin de pouvoir estimer la consommation énergétique des phases d'apprentissage et d'inférence (puisque c'est ce qui consommera le plus après).

Enfin, il reste intéressant de noter que ces réseaux s'utilisent rarement seuls (architectures hybrides comme par exemple les CNN - LSTM avec un CNN en entrée, un LSTM au milieu et un MLP en sortie).

b. Analyse de la littérature et des bonnes pratiques pour CNN

Afin de mieux interpréter les résultats obtenus et d'affiner nos propos, nous avons décidé de nous baser sur une étude intitulée "*ML Estimation of energy consumption in machine learning*" de Garcia Martin, qui détaille les différentes méthodes existantes d'estimation de la consommation énergétique puis teste certaines de ces méthodes sur des modèles de CNN (3 types différents), afin de déterminer celui consommant le moins d'énergie.²¹

Ainsi, d'après l'étude, les premiers modèles d'estimation de l'énergie comptaient les opérations de multiplication-accumulation (**MAC**) pour évaluer le nombre d'opérations en virgule flottante et le nombre d'accès à la mémoire, servant de proxy pour l'énergie consommée. Des techniques telles que le pruning, la compression et les modèles compacts ont été utilisées pour réduire le nombre de paramètres des réseaux neuronaux, diminuant ainsi la consommation d'énergie.

Les approches plus récentes intègrent le coût énergétique des différentes mémoires hiérarchiques. Un modèle d'estimation a été développé en comptant le nombre de réutilisation de chaque valeur de donnée à travers les différents niveaux de mémoire (DRAM, buffer global, array, et registre), avec des coûts énergétiques spécifiques pour chaque niveau. Ce modèle a été appliqué à des accélérateurs spécifiques comme Eyeriss.

Des modèles d'estimation énergétique au niveau applicatif utilisant des compteurs de performance pour les processeurs généraux ont également émergé. **SyNERGY**, par exemple, utilise des compteurs de performance pour estimer l'énergie consommée par des réseaux neuronaux convolutifs sur un CPU ARM A57, en prenant en compte les instructions SIMD et les accès mémoire principaux.

NeuralPower, quant à lui, utilise une approche de régression pour modéliser la consommation d'énergie et le temps d'exécution sur un GPU de bureau. Les modèles sont construits pour les principales couches d'un réseau neuronal convolutif, utilisant des valeurs réelles de puissance et de temps. Les caractéristiques du modèle, comme la taille du kernel et le nombre de couches, sont utilisées pour estimer la consommation d'énergie de chaque couche.

DeLight modélise la consommation d'énergie d'un réseau neuronal simple lors de la phase d'entraînement. L'énergie est modélisée en termes d'opérations arithmétiques de base et de communication des poids partagés dans un environnement d'entraînement distribué. Les coefficients de modélisation sont obtenus via des micro-benchmarks sur les cœurs CUDA de la plateforme Nvidia TK1.

L'**inférence d'un réseau neuronal convolutionnel** (ConvNet) consiste à passer une image à travers un ConvNet pré-entraîné pour en extraire des caractéristiques utiles pour des tâches comme la classification d'images. Cette inférence est souvent réalisée sur des systèmes mobiles ou embarqués avec une durée de vie de batterie limitée. Les objectifs sont d'utiliser une méthodologie existante pour obtenir l'énergie par couche d'un

21. Nous nous sommes intéressées au CNN car des 3 codes, il était celui avec la meilleure précision.

ConvNet, d'appliquer une approche basée sur la régression aux informations des compteurs de performance pour prédire l'énergie de la couche convolutionnelle, et de valider avec des mesures réelles d'énergie. La méthodologie utilisée, SyNERGY, implique l'insertion d'annotations de code dans le code C++ de Caffe et la capture de la puissance avec un capteur sur le Jetson TX1.

Après avoir donc présenté ces différentes méthodes, les chercheurs ont effectué des tests sur ces 3 types de CNN : Inception-v3, MobileNet et DenseNet. Leurs résultats montrent que la prédiction d'énergie et les mesures réelles pour les couches convolutionnelles de MobileNet sont les plus basses, faisant de **MobileNet le choix le plus économe en énergie parmi les trois, avec une précision relative moyenne de 68,91%** (García-Martín et al. [2019]). MobileNet est donc recommandé pour des environnements d'exécution similaires en termes d'efficacité énergétique, notamment en embarqué.

Une autre conclusion que nous pouvons en tirer est qu'au sein même des grandes catégories d'algorithme de DL, il faut s'attarder sur les algorithmes existants afin de faire un choix adapté.

c. Implémentation d'algorithmes d'IA générative

Nous nous sommes ensuite intéressées aux réseaux GAN et Transformer qui permettent tous les deux de générer des données. L'exemple de Transformer est particulièrement intéressant puisqu'il correspond, comme sus-mentionné, à l'algorithme à l'origine de ChatGPT, massivement utilisé aujourd'hui et consommant une énorme quantité d'énergie, puisque pour rappel, les IA génératives consomment beaucoup plus lors de la phase d'inférence.

Pour créer le GAN, nous nous sommes aidées de la documentation officielle de TensorFlow. Nous n'avons malheureusement pas réussi à faire tourner notre programme (Annex 3) au delà d'une epoch (arrêt de Kernel), ce qui peut probablement s'expliquer par la non-utilisation d'un GPU (ce que nous n'avons pas à disposition).

De même, nous nous sommes aidées de la documentation officielle de TensorFlow pour créer notre Transformer mais le test d'a pu pu aboutir car nous n'avons pas réussi à faire tourner notre programme (nous avons laissé le code de ce dernier dans la branche "essais" de notre GitHub). Nous ne savons pas ici si le problème provient toujours du GPU ou s'il est autre. C'est pourquoi, nous préconisons d'ailleurs de disposer de matériel performant pour permettre la réalisation de tels tests et afin d'obtenir des résultats exploitables lors de futurs projets d'étude.

Ce type de réseaux nécessitent donc d'utiliser des processeurs plus puissants, avec une empreinte carbone élevée. Il faudrait donc trouver comment réduire la consommation d'énergie au sein des architectures et infrastructures.

2.2 Pour réduire la consommation d'énergie au sein des architectures et infrastructures

Au delà de l'optimisation des algorithmes de calcul, il est également indispensable de réaliser des économies d'énergie sur les infrastructures informatiques elles-mêmes.

2.2.1 Au sein des architectures

Comme mentionné dans la première grande partie, l'architecture des systèmes informatiques joue un rôle majeur dans la réduction de leur consommation énergétique. Parmi les stratégies à l'étude, nous pouvons mentionner : (i) *la virtualisation*, qui permet de maximiser l'utilisation des ressources physiques disponibles, réduisant ainsi le nombre de serveurs nécessaires, et *la conteneurisation*, par exemple via Docker, qui permet également d'optimiser l'utilisation des ressources ; (ii) les architectures nuagique ou le *Cloud Computing*²², qui permettent de réaliser des économies d'échelle et d'optimiser la consommation énergétique par les fournisseurs de services cloud (Apirajitha. and Angayarkanni [2012]) ; (iii) *l'Edge Computing*, qui permet d'entraîner des algorithmes d'apprentissage sur des données à la périphérie du réseau, sans les transférer, plutôt que de manière centralisé ; ou encore (iv) *les architectures neuromorphiques*, qui vont nous intéresser tout particulièrement ici.

L'informatique neuromorphique ou **neuromorphic computing** voit le jour avec Carver Mead, à la fin des années 1980, en tant que "*paradigme de conception analogique/signal mixte permettant d'imiter efficacement les fonctions neurophysiologiques dans les circuits intégrés (CI) en tirant parti de la richesse physique des transistors et d'autres dispositifs électroniques*" (Hendy and Merkel [2022]). Aujourd'hui, l'informatique neuromorphique désigne à la fois, un paradigme pour la conception de matériel (hardware), et pour la conception d'algorithmes (software).

Les **architectures neuromorphiques** s'inspirent fortement du fonctionnement du système nerveux humain. En effet, alors que les architectures statiques, comme celles de Von Neumann, sont basées sur une logique booléenne, les neurones, dont les synapses changent dynamiquement, constituent les blocs fondamentaux des architectures neuromorphiques. Les ordinateurs de Von Neumann, majoritaires aujourd'hui, représentent l'information en format binaire de 32 ou 64 bits, via un matériel numérique déterministe. A l'inverse, notre cerveau représente l'information biologique sous forme de potentiels d'action (aussi appelés « spikes ») émis par des neurones. Les systèmes neuromorphiques vont s'inspirer de ce principe de traitement parallèle et hautement asynchrone de l'information. Dernière différence majeure entre ces deux architectures, la mémoire et les unités de traitement sont étroitement couplées dans un système neuromorphique, alors qu'elles sont séparées physiquement dans un calculateur de Von Neumann (Nawrocki et al. [2016]).

22. Le cloud computing permet aux utilisateurs d'accéder via internet à des ressources informatiques et à des applications hébergées sur des serveurs distants plutôt que sur leurs propres machines locales.

L'avantage d'une approche neuromorphique est que la consommation d'énergie globale est très faible. Cela est dû au fait que les neurones, sollicités de manière asynchrone, ne participent pas à tous les calculs et ne consomment donc pratiquement pas d'énergie (Stromatias [2020]). En effet, un des problèmes majeurs aujourd'hui en recherche est que les chercheurs cherchent à obtenir de nombreux résultats, rapidement. Pour cela, ils utilisent des unités de traitement graphiques ou **GPU**, qui permettent d'accélérer le processus d'expérimentation. Ces puces peuvent effectuer des multiplications et des convolutions matricielles très efficaces (soient la grande majorité des opérations mathématiques utilisées lors de l'apprentissage profond).

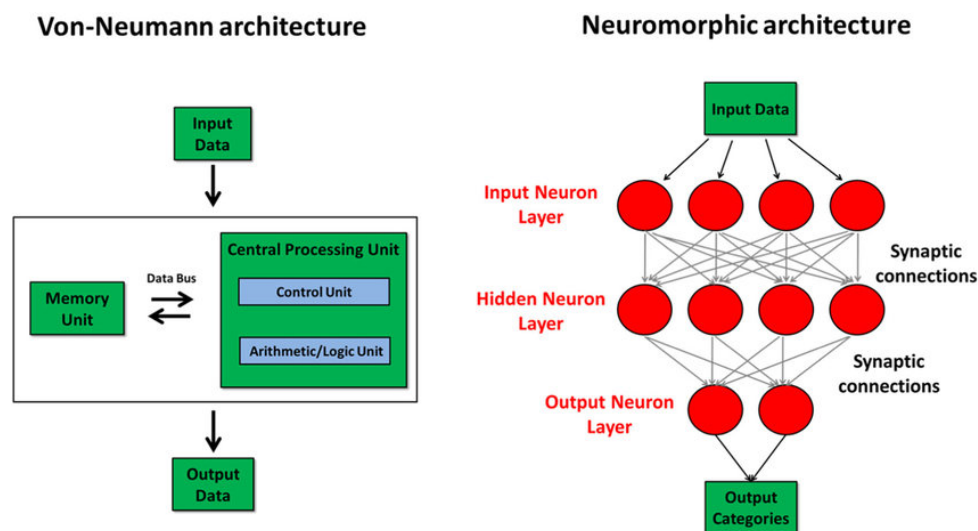


FIGURE 14 – Comparaison des architectures Von Neumann et des architectures neuromorphiques. Source : Del Valle et al. [2018].

Or, les GPU « *dissipent de trop grandes quantités d'énergie.* » Les GPU NVIDIA par exemple (ici, le modèle en question est le 512 GPU NVIDIA V100), sont parmi les plus répandus sur le marché, et ont été utilisés par Shoybi *et al.* en 2019, pour former un réseau très profond à même d'effectuer des tâches de traitement du langage naturel. "Ce modèle avait 8,3 milliards de paramètres entraînaables. Sa formation a nécessité 9,2 jours d'utilisation continue des GPU. Étant donné que chaque GPU V100 a une dissipation d'énergie maximale de 250 W, l'énergie totale nécessaire à l'apprentissage de ce modèle a été deux fois supérieure à l'énergie moyenne consommée par un ménage américain au cours d'une année." Cette consommation énergétique est astronomique. L'utilisation de ces GPU est également problématique pour une **démocratisation de la recherche** en DL, puisque les coûts énergétiques et la quantité de ressources (mémoire, calculs) associées à l'entraînement du réseau rend l'utilisation de ces cartes très limitées. Seules les grandes entreprises peuvent y accéder (Shoybi et al. [2019]).

Mais en quoi consiste concrètement l'informatique neuromorphique ? Comme mentionné plus haut, elle comprend à la fois des algorithmes neuromorphiques, appelés réseaux de neurones à pointes (*Spiking Neural Networks* ou **SNN**), et du matériel neuromorphique, c'est-à-dire "des ASIC dédiés et optimisés pour les SNN".

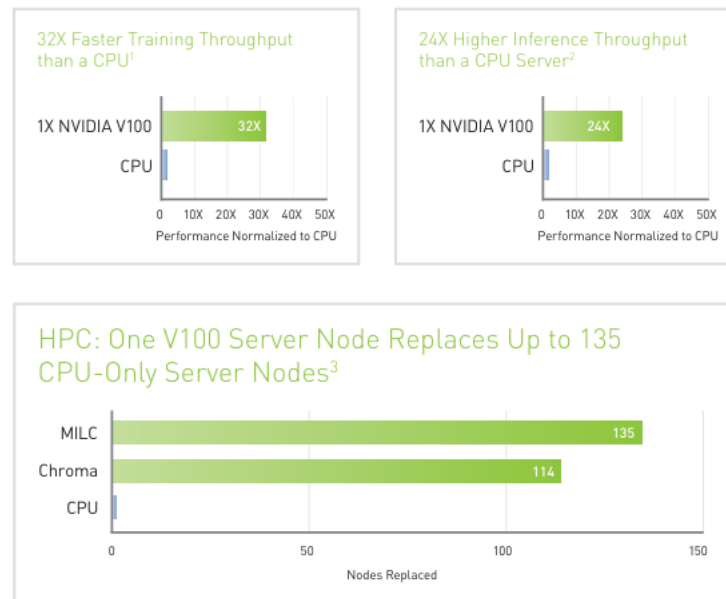


FIGURE 15 – Comparaison des performances du GPU V100 face à un CPU classique : "A single V100 Tensor Core GPU offers the performance of nearly 32 CPUs". Source : fiche technique du GPU NVIDIA V100.

Même si les équations mathématiques décrivant leur fonctionnement datent des années 1950 (Hodgkin et Huxley, Nobel 1952), on considère que les SNN constituent la troisième génération de réseaux neuronaux artificiels. Les pointes (représentées par des « 0 » et « 1 » dans le système informatique, où « 0 » signifie l'absence de pointe) sont utilisées pour transmettre des informations entre neurones. Avec un tel mécanisme, *"les multiplications coûteuses peuvent être remplacées par des additions plus économes en énergie, ce qui atténue l'intensité du calcul."* Le calcul dans les réseaux neuronaux SNN est donc basé sur les événements, comme dans le système nerveux, de sorte que chaque neurone du réseau ne génère ses sorties que lorsqu'un nombre suffisant de pics indiquant l'existence d'une caractéristique a été détecté. Cette caractéristique permet aux réseaux neuronaux SNN de résoudre des tâches spatio-temporelles complexes et d'utiliser des capteurs efficaces basés sur les événements, tels que les caméras basées sur les événements. Ainsi, les SNN, contrairement aux ANN, introduisent la notion de temps (Luo et al. [2023]).

Au niveau matériel, les calculs sont intégrés dans une architecture de mémoire distribuée. L'énergie statique et l'énergie consommée sont réduites grâce à la combinaison de différentes mémoires (RRAM et MRAM qui sont résistives et magnéto-résistives non volatiles).

L'étude *"Are SNNs Really More Energy-Efficient Than ANNs? an In-Depth Hardware-Aware Study"* (Dampfhofer et al. [2023]) compare des réseaux SNN et ANN sur des plateformes matérielles spécifiques (hardware), tout en prenant en compte les accès mémoires. Les SNN se révèlent plus efficaces énergétiquement, surtout à faible densité de pics (ou *spike* en anglais), mais l'efficacité de ces réseaux varie en fonction des spécificités du matériel et de la concentration (ou *sparsity*) des pics. On obtient par ex. les résultats suivants :

<i>Sparsité des pics</i>	<i>Efficacité du réseau ?</i>
0.1	SNN 3.6x > ANN
0.05	SNN 7.3x > ANN

TABLEAU 17 – Résultats de l'étude Dampfhofer et al. [2023].

L'informatique neuromorphique possède cependant plusieurs limites : les algorithmes SNN ne peuvent pas utiliser directement les algorithmes d'apprentissage ANN comme la **rétropropagation**, ce qui les rend moins populaires. En effet, les algorithmes de rétropropagation sont basés sur des équations différentielles incompatibles avec les équations des SNN, qui sont *"discontinues en raison de la fonction de seuillage de la membrane"*. De plus, comme rapidement évoqué dans l'étude précédente, les architectures processeurs ou encore GPU actuelles ne sont pas adaptées à la mise en œuvre efficace de SNN, puisque ces derniers fonctionnent *"de manière asynchrone, ont un parallélisme massif, une communication éparse et un calcul en mémoire, ce qui contraste fortement avec les opérations séquentielles et la séparation de von Neumann entre la mémoire et le calcul sur les CPU et les GPU"*. La recherche sur ce sujet est cependant en plein essor.

Il existe d'ailleurs des API permettant de tester des architectures neuromorphiques sur ses programmes personnels, ce qui peut pousser plus de programmeurs et ingénieurs à s'intéresser à ces questions et à comparer la consommation énergétique de ces différentes architectures. L'organisation *Open Neuromorphic*²³ (ou ONM), présente sur GitHub, propose à la fois une liste de frameworks logiciels (pour trouver l'outil adapté à nos besoins) et une plate-forme pour nos programmes.

Alors que nous cherchions comment tester nos différents programmes sur différents types d'architectures, nous avons également découvert des **bioprocresseurs**, développés par l'entreprise suisse **FinalSpark** (cf. Figure 16). Le concept de bioprocresseur repose sur l'utilisation de *"neurones vivants pour effectuer des calculs, de la même manière que les réseaux neuronaux artificiels sont utilisés aujourd'hui"*. Selon cette société, l'informatique biologique ou **biocomputing** pourrait être *"plus rapide, plus efficace et plus puissante que l'informatique et l'IA basées sur le silicium, et ne nécessiter qu'une fraction de l'énergie"* (Jordan et al. [2024]).

Dans un de leur articles (Smirnova et al. [2023]), ils s'intéressent à des systèmes bio-informatiques basés sur les organoïdes²⁴, qui pourraient permettre une prise de décision plus rapide et une meilleure efficacité énergétique. Cet article ne fournit cependant pas de résultats chiffrés précis à l'appui de l'efficacité énergétique réelle de cet outil. Après plusieurs échanges avec eux (au cours desquels nous leur avons envoyés notre lien GitHub avec les programmes de ML et de DL que nous souhaitions tester), nous n'avons finalement pas pu tester nos programmes sur la *neuroplatform* FinalSpark, puisque nos codes avaient

23. Lien GitHub : <https://github.com/open-neuromorphic/open-neuromorphic>

24. Les organoïdes sont des versions miniatures et simplifiées d'un organe, fabriquées in vitro en trois dimensions et qui présentent une micro-anatomie réaliste. (source : Wikipedia)

initialement été "conçus pour tester la consommation d'énergie de l'informatique *in silico*". Cela serait néanmoins envisageable dans le cadre d'un projet plus approfondi permettant l'élaboration de codes complexes.

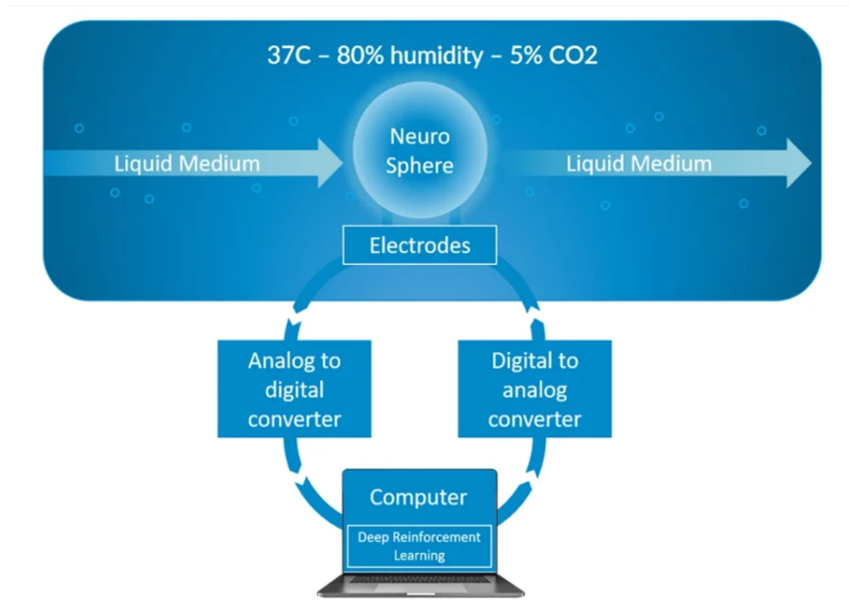


FIGURE 16 – Fonctionnement d'un bioprocasseur. Source : site officiel de FinalSpark.

2.2.2 Dans les data centers

Comme nous l'avons vu précédemment, les calculs de la consommation énergétique liée à l'IA prennent souvent en compte la consommation des data centers. Leur contribution est non négligeable et dans le domaine du numérique, ils font partie des plus gros consommateurs d'énergie. Pour réduire leur empreinte écologique, plusieurs mesures peuvent être mises en place.

Les recherches les plus anciennes ont exploré l'intégration des centres de données avec la réponse à la demande et le déplacement des charges pour réduire les coûts d'électricité. Cependant, ces approches supposent souvent une collaboration avec les gestionnaires de système indépendants (ISO) ou utilisent des métriques simplifiées des émissions de carbone. Dans une étude de Lindberg et al. [2021], il a été montré que la flexibilité des centres de données est bénéfique pour réduire les émissions de carbone et les coûts de génération par rapport à un modèle sans flexibilité.

Utilisation de sources d'énergie moins polluantes

L'introduction de carburants biosourcés, comme le HVO100 (*Hydrotreated Vegetable Oil*) de TotalEnergies, peut permettre une réduction significative des émissions de GES. Ce biocarburant, produit à partir d'huiles végétales et de graisses animales, peut réduire

les émissions de CO₂ de 70% à 80% par rapport aux carburants fossiles traditionnels. Les centres de données peuvent également bénéficier de l'installation de panneaux solaires, d'éoliennes ou d'autres sources d'énergie renouvelable. Par exemple, un data center de Google à Eemshaven, aux Pays-Bas, fonctionne principalement à l'énergie éolienne, ce qui contribue à l'objectif de Google de fonctionner entièrement sur des énergies renouvelables d'ici 2030.

Amélioration du PUE

Le PUE (*Power Usage Effectiveness*) est un indicateur utilisé pour qualifier l'efficacité énergétique d'un centre d'exploitation informatique :

$$PUE = \frac{\text{Energie totale consommée par le centre}}{\text{Energie consommée par les équipements informatiques}}$$

Afin de minimiser le PUE, on peut notamment privilégier le refroidissement à l'eau plutôt qu'à l'air. Ainsi, le data center **Lefdal Mine**, situé en Norvège, est l'un des centres de données les plus efficaces d'Europe. Il est installé dans une mine profonde qui le protège de l'ensoleillement et des conditions climatiques extrêmes. Le centre de données utilise de l'énergie provenant de barrages hydroélectriques et se compose de six niveaux souterrains divisés en 75 chambres, offrant un espace blanc potentiel de 120 000 m². La solution de refroidissement de Lefdal Mine²⁵, basée sur la circulation d'eau de mer froide à travers un échangeur de chaleur, permet d'atteindre un PUE de 1,08 à 1,15. Moins de 3% de l'énergie dépensée pour les technologies de l'information est utilisée pour le refroidissement, avec une configuration de 5 kW par rack. L'eau de mer utilisée pour le refroidissement maintient une température constante de 8 °C toute l'année (Gül Nihal Gugul and Sakir Tasdemir [2022]).

La récupération de la chaleur est également une solution afin de diminuer le PUE. Elle est d'ailleurs fortement liée au refroidissement car la plupart de ces centres utilisent un refroidissement par air qui rendent difficile la capture et le transport de la chaleur qui se situe entre 15 et 45 °C. La réutilisation de la chaleur est une option intéressante, notamment en termes de chauffage urbain, qui promet donc des avantages non négligeables. En effet, une étude a pu montrer que jusqu'à 55,4% des besoins de chauffage de certains quartiers de Londres pourraient être couverts par la chaleur excédentaire des centres de données voisins (Davies et al. [2016]).

Amélioration du WUE

L'amélioration de l'efficacité du *Water Usage Effectiveness* (WUE) dans les data centers peut se réaliser de plusieurs manières. Tout d'abord, la dissipation de la chaleur via des échangeurs air/eau secs permet de réduire la consommation d'eau en utilisant des systèmes qui refroidissent l'air sans évaporation directe d'eau. Cette méthode est particulièrement efficace dans les climats secs où la demande en refroidissement est élevée. De plus, l'utilisation de la brumisation lorsque la température extérieure dépasse 25°C offre une solution efficace pour abaisser la température de l'air ambiant sans une consommation

25. <https://www.lefdalmine.com>

excessive d'eau. Enfin, l'intégration de médias humides pour recycler l'eau non-évaporée permet de réduire jusqu'à 50% la consommation d'eau globale en maintenant un environnement de refroidissement efficace tout en minimisant les pertes.

Déplacement dans le temps et dans l'espace

Pour réduire les émissions de carbone, des stratégies de déplacement des charges de travail dans le temps et dans l'espace sont utilisées. Le déplacement dans le temps consiste à répartir dynamiquement les charges de travail pour éviter les surcharges sur certains serveurs et permettre à d'autres de fonctionner en mode basse consommation ou d'être éteints lorsqu'ils ne sont pas utilisés. Le déplacement dans l'espace permet aux entreprises de différer ou de déplacer géographiquement leurs tâches informatiques, adaptant ainsi leur consommation d'électricité. Par exemple, exécuter un code dans un data center en Australie peut coûter 147 fois plus en terme de CO₂ que l'exécuter en Suède (cf. Figure 17). Google utilise cette approche en déplaçant les tâches informatiques vers des heures ou des lieux moins intensifs en production de carbone (Lindberg et al. [2021]).

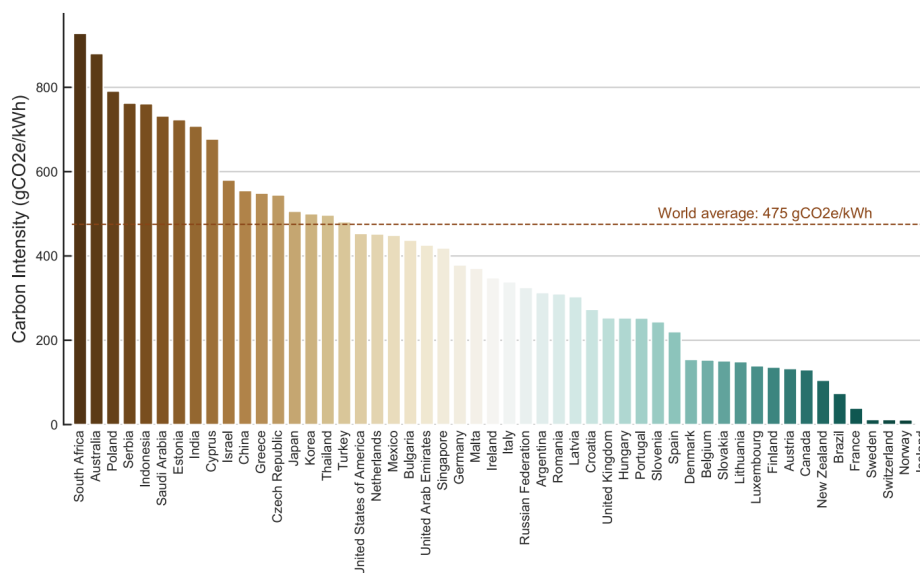


FIGURE 17 – Intensité carbone des data centers selon leur localisation. Source : Green Algorithms.

Autres leviers d'actions

D'autres mesures qui peuvent sembler moins directement liées au fonctionnement même des data centers sont pourtant primordiales. Ainsi, il est essentiel de limiter le recours au fret aérien pour le transport des équipements informatiques et privilégier les options terrestres et maritimes à faible impact carbone. Cela inclut l'utilisation de camions et de navires de transport équipés de technologies écoénergétiques. De plus, encourager le télétravail dans le secteur des data centers réduit non seulement les déplacements liés aux

réunions et aux formations, mais aussi les déplacements domicile-travail des employés. Pour les déplacements professionnels nécessaires, favoriser le train pour les courtes distances est une option écologique par rapport aux vols courts.

En tenant compte des principales sources de pollution développées dans la partie 1., il nous est possible de catégoriser les leviers d'actions en 3 groupes selon les catégories d'émissions de gaz à effet de serre utilisées pour réaliser un bilan carbone (résumés dans le tableau 18). Nous avons choisi de nous baser sur les objectifs d'émission carbone qui ont été introduites pour la première fois par le *Greenhouse Gas Protocol*²⁶. Il s'agit d'une norme internationale de calcul de l'empreinte carbone des entreprises et des organisations créée à la fin des années 90 par le *World Resources Institute* et le *World Business Council for Sustainable Development*.

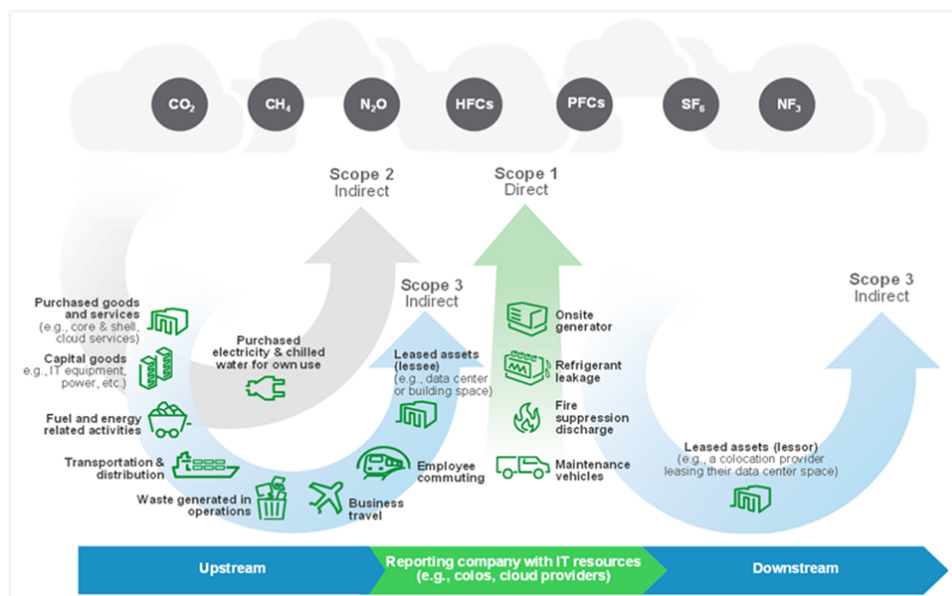


FIGURE 18 – Objectifs ciblés des data centers utilisés afin de définir les leviers d'actions.
Source : Schneider Electric.

L'objectif 1 englobe les **émissions directes** résultant de la combustion d'énergies fossiles. L'objectif 2, quant à lui, concerne les émissions indirectes de gaz à effet de serre, principalement associées à la **consommation d'électricité** nécessaire au fonctionnement des équipements informatiques. Enfin, l'objectif 3 inclut les émissions indirectes de gaz à effet de serre liées à la **fabrication** (comme mentionné précédemment) et au **transport des équipements informatiques**, ainsi qu'à la gestion des déchets électroniques.

26. <https://ghgprotocol.org>

Objectif 1	Objectif 2	Objectif 3
Utilisation de carburants verts et renouvelables	Amélioration du PUE	Optimisation des frets/-voyages
Refroidissement à l'eau	Décarbonation du mix énergétique	Augmentation de la durée de vie des serveurs
Mutualisation des environnements physiques de production	Introduction de labels énergétiques	Utilisation de matériel reconditionné
Maintenance régulière des équipements	Optimisation de la consommation en fonction des besoins réels	Virtualisation des serveurs

TABLEAU 18 – Récapitulatif des leviers d'actions par objectif d'après Grégory Lebourg [2024].

Les centres de données, en tant que gros consommateurs d'énergie, jouent un rôle crucial dans la réduction de l'empreinte écologique globale. Une combinaison de mesures directes et indirectes, soutenue par une gestion intelligente des charges de travail, peut significativement réduire l'empreinte carbone des centres de données. Ces efforts, lorsqu'ils sont mis en œuvre de manière coordonnée et stratégique, permettent non seulement de répondre aux défis environnementaux actuels, mais aussi de préparer un avenir plus durable pour l'industrie informatique.

Focus sur la stratégie des 4M

Beaucoup d'articles proposent des solutions pour réduire l'empreinte carbone de l'IA. Tout au long de ce rapport, nous en avons proposés également. Nous avons décidé de faire un focus sur une des stratégies qui revient souvent et qui offre des possibilités d'améliorations à la fois de l'algorithmique (cf. Partie 2.1.), des architectures (cf. Partie 2.2.A.) et des data centers : les "4M".

Google a proposé 4 axes, appelés les "4M" (Model, Machine, Mechanization, Map Optimization), permettant de réduire les coûts liés à l'entraînement des modèles de ML et DL (Patterson et al. [2022]).

- Model : La sélection de **modèles d'apprentissage automatique plus efficaces**, tels que les modèles sparse²⁷ par rapport aux modèles denses²⁸, peut réduire le calcul nécessaire par un facteur de 5 à 10 tout en maintenant la qualité du ML.

- Machine : L'utilisation de **processeurs optimisés** pour l'entraînement des modèles ML, comme les unités de traitement tensoriel (TPU) et les GPU récents (par exemple, les V100 et A100), par rapport aux processeurs généralistes, peut améliorer les performances par watt par un facteur de 2 à 5.

27. Seuls certains poids importants sont utilisés.

28. Tous les poids du réseau sont utilisés, connexions complètes entre les neurones de chaque couche.

- Mechanization : Le **calcul dans le cloud**, plutôt que sur site, améliore l'efficacité énergétique des centres de données, réduisant les coûts énergétiques par un facteur de 1,4 à 2. Les centres de données dans le cloud sont spécialement conçus pour l'efficacité énergétique, contrairement aux centres de données sur site souvent situés dans des espaces plus anciens et moins efficaces.

- Map : Le cloud-computing permet aux praticiens du ML de choisir des **emplacements géographiques avec une énergie plus propre**, réduisant davantage l'empreinte carbone brute par un facteur de 5 à 10. En utilisant des clouds neutres en carbone (souvent par compensation des émissions émises) l'empreinte peut en théorie être réduite à zéro.

En combinant ces quatre stratégies, il serait possible de réduire la consommation d'énergie d'un facteur 83 et les émissions de dioxyde de carbone de 747 fois sur quatre ans, tout en maintenant un même niveau de qualité.

★ ★ ★

3 - Critiques

Nous n'avons pas évoqué les critiques qui ont pu être formulées au sujet du *Green AI* et du *Green Computing*. En effet, nous ne pouvons ignorer l'existence d'effets rebonds. Pour rappel, en écologie, les effets rebonds *"caractérisent un effet pervers et paradoxal des progrès en matière d'efficacité énergétique : les économies réalisées ne sont pas synonymes d'une moindre consommation, mais entraînent au contraire une augmentation de la consommation des équipements concernés ou d'autres équipements, et donc de l'énergie nécessaire à leur fabrication et à leur fonctionnement."* Dans leur article intitulé "Towards Sustainable Artificial Intelligence : An Overview of Environmental Protection Uses and Issues" (OpenStudio, France et al. [2023]), Arnault Pachot et Céline Patissier mettent en avant ce point : les solutions visant une amélioration de l'impact environnemental de l'IA, peuvent elles-mêmes conduire à une augmentation de la complexité des modèles d'IA et par conséquent à une augmentation de leur consommation énergétique globale. En outre, de nombreuses entreprises essayent maintenant de mettre en avant leur transition vers une « IA verte ». Certains les accusent de Greenwashing. Néanmoins, si le Greenwashing peut parfois être une façon de faire un premier pas vers une transition écologique aboutie, ce n'est souvent qu'un moyen bref et trompeur de donner une image écologique sans réel engagement en profondeur pour l'environnement.

De plus, nous avons observé un manque de recommandations officielles et de réglementations à grande échelle, qu'elles soient internationales, continentales ou nationales. Aucun texte de loi sur l'IA n'aborde l'aspect écologique de manière approfondie. En France, l'idée de certificats de sobriété numérique a été proposée pour encourager les entreprises à adopter des pratiques plus écologiques (Nicolai and Peragin [2022]). Toutefois, ces initiatives restent limitées et n'ont pas encore été adoptées de manière concrète au niveau gouvernemental. L'Union Européenne, quant à elle, travaille sur des directives pour rendre les technologies plus durables, mais ces initiatives n'ont pas encore abouti à des réglementations strictes sur l'IA en particulier.

Enfin, dans le cadre de cette TX, il s'est avéré assez difficile pour nous de trouver des informations chiffrées. En effet, la plupart des articles, présentant par exemple des architectures innovantes, évoquent une baisse de la consommation et un ordre de grandeur mais donnent très rarement des données précises. Nous avons deux explications à cela qui dépendent l'une de l'autre. Premièrement, un des constats que nous avons fait est qu'il n'existe pas vraiment de calcul faisant l'unanimité et utilisé/utilisable par la majorité des chercheurs et chercheuses. Ainsi, tous les outils que nous avons utilisés et les articles que nous avons lus précisaient leur propre manière d'effectuer les calculs de coûts énergétique ou d'empreinte carbone. Bien que ces calculs soient souvent similaires, cela ne permet pas une comparaison parfaite. Deuxièmement, en effectuant nos recherches, il nous a semblé que le manque d'évaluations des algorithmes en termes de consommation énergétique peut être attribué au manque d'outils appropriés pour effectuer ces mesures. Nous l'avons expérimenté en voulant tester nos codes : une majorité d'outils a été développé pour une utilisation dans un environnement Linux (et non sur Windows ou MacOS davantage utilisés par la population, comme par un grand nombre de programmeurs).

Ils fonctionnent souvent uniquement pour un seul langage de programmation et ne sont utilisables que sur des GPU NVIDIA. L'outil RAPL, par exemple, ne s'utilise que sur Linux et lorsque nous avons essayé de tester nos codes sur des machines Linux de l'UTC, nous nous sommes rapidement trouvées bloquées par une limite d'espace mémoire. Le manque d'outils compatibles et faciles d'utilisation nous a fortement limité lors de nos tests. Nous nous sommes donc tournées vers CodeCarbon, mais nous aurions bien aimé tester d'autres outils afin de pouvoir comparer leurs mesures et rendus.

★ ★ ★

Conclusion

Résumé des bonnes pratiques

Plusieurs travaux de recherches menés par des universités ou des entreprises ont permis d'établir des recommandations afin d'utiliser l'informatique de manière plus durable. En rédigeant notre rapport, nous en avons sélectionné certaines qui nous semblaient pertinentes. Les recommandations de l'article "*Ten simple rules to make your computing more environmentally sustainable*" (Lannelongue et al. [2021]) résument bien l'ensemble de nos recherches, c'est pourquoi nous les présentons ci-dessous en guise de synthèse.

1. Calculer l'empreinte carbone du travail.
2. Inclure l'empreinte carbone dans l'analyse coût-bénéfice.
3. Conserver, réparer et réutiliser les appareils pour minimiser les déchets électroniques.
4. Choisir soigneusement le centre de calcul.
5. Choisir soigneusement le matériel.
6. Augmenter l'efficacité du code.
7. Être un analyste frugal.
8. Rendre claires ses exigences matérielles et son empreinte carbone.
9. Être conscient des conséquences inattendues de l'amélioration de l'efficacité des logiciels.
10. Compenser son empreinte carbone.

Ces bonnes pratiques sont utiles mais nous semblent insuffisantes en 2024 et mériteraient d'être un peu plus spécifiées, c'est pourquoi nous proposons d'ajouter également :

- Intégrer des tests systématiques pour mesurer l'empreinte carbone des codes dans les entreprises et les universités.

- Privilégier certains algorithmes en tenant compte du compromis entre précision et consommation.

- Rester prudent face aux effets de mode de certains langages plus consommateurs et créer des librairies sur des langages moins consommateurs²⁹.

29. Les quelques API disponibles sur GitHub pour utiliser Tensor Flow avec C++ sont, par exemple, difficiles à installer. Il faudrait démocratiser l'accès, sur des langages compilés, aux libraires de ML et DL.

- Sensibiliser les étudiants en informatique aux enjeux écologiques et aux bonnes pratiques liés au numérique.
- Instaurer des réglementations officielles et inclure ces enjeux dans les textes législatifs relatifs à l'IA.

Bilan

En conclusion, la réduction de l'empreinte carbone de l'IA nécessite une approche qui englobe tous les aspects techniques tels que l'algorithmique mais également les architectures et le matériel. Le concept de Green AI met en lumière cette nécessité de concevoir des solutions d'IA non seulement performantes, mais aussi respectueuses de l'environnement. En nous concentrant principalement sur les spécificités techniques, nous n'avons pas traité les problèmes éthiques associés, qui sont néanmoins fondamentaux. Ces considérations sont essentielles afin de permettre une réduction saine de l'empreinte carbone et une transition écologique aboutie dans le domaine du numérique. En effet, l'explosion de l'IA depuis quelques années engendre des défis majeurs, allant de la protection de la vie privée, à la lutte contre la discrimination algorithmique ou à la responsabilité des décisions prises par les systèmes d'IA. Intégrer des principes éthiques dans la conception et le déploiement de technologies d'IA verte nous semble donc tout aussi indispensable pour garantir un avenir durable pour notre planète.

Références

- Payal Dhar. The carbon impact of artificial intelligence. *Nature Machine Intelligence*, 2 (8) :423–425, August 2020. ISSN 2522-5839. doi : 10.1038/s42256-020-0219-9. URL <https://www.nature.com/articles/s42256-020-0219-9>. Publisher : Nature Publishing Group. 1
- Arcep. Etude ADEME – Arcep sur l’empreinte environnementale du numérique en 2020, 2030 et 2050, March 2023. URL <https://www.arcep.fr/la-regulation/grands-dossiers-thematiques-transverses/lempreinte-environnementale-du-numerique/etude-ademe-arcep-empreinte-environnemental-numerique-2020-2030-2050.html>. 1
- Grégory Lebourg. L’impact environnemental du Cloud : Bilan et Solutions, April 2024. 3, 4, 5, 43, i, iii
- Sébastien Broca. Le numérique carbure au charbon - Le Monde diplomatique. 2020. 5
- Mueen Uddin, Yasaman Darabidarabkhani, Asadullah Shah, and Jamshed Memon. Evaluating power efficient algorithms for efficiency and carbon emissions in cloud data centers : A review. *Renewable and Sustainable Energy Reviews*, 51 :1553–1563, November 2015. ISSN 13640321. doi : 10.1016/j.rser.2015.07.061. URL <https://linkinghub.elsevier.com/retrieve/pii/S136403211500708X>. 6
- Arman Shehabi, Eric Masanet, Hillary Price, Arpad Horvath, and William W. Nazaroff. Data center design and location : Consequences for electricity use and greenhouse-gas emissions. *Building and Environment*, 46(5) :990–998, May 2011. ISSN 03601323. doi : 10.1016/j.buildenv.2010.10.023. URL <https://linkinghub.elsevier.com/retrieve/pii/S036013231000315X>. 6
- Mark Horowitz. 1.1 Computing’s energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, San Francisco, CA, USA, February 2014. IEEE. ISBN 978-1-4799-0920-9 978-1-4799-0918-6. doi : 10.1109/ISSCC.2014.6757323. URL <http://ieeexplore.ieee.org/document/6757323/>. 10
- John L. Hennessy and Patterson David A. *Computer Architecture : A Quantitative Approach*. 2011. 10
- Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784) :607–617, November 2019. ISSN 0028-0836, 1476-4687. doi : 10.1038/s41586-019-1677-2. URL <https://www.nature.com/articles/s41586-019-1677-2>. 10
- L. Prechelt. An empirical comparison of seven programming languages. *Computer*, 33 (10) :23–29, October 2000. ISSN 00189162. doi : 10.1109/2.876288. URL <http://ieeexplore.ieee.org/document/876288/>. 11

- Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. Energy efficiency across programming languages : how do energy, time, and memory relate? In Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, pages 256–267, Vancouver BC Canada, October 2017. ACM. ISBN 978-1-4503-5525-4. doi : 10.1145/3136014.3136031. URL <https://dl.acm.org/doi/10.1145/3136014.3136031>. 12
- Jean-paul Barthès. Architecture d’une interface conversationnelle pour les agents assistants personnels. January 2004. URL https://www.academia.edu/83945726/Architecture_d_une_interface_conversationnelle_pour_les_agents_assistants_personnels. 17
- Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model, 2022. URL <https://arxiv.org/abs/2211.02001>. Version Number : 1. 17
- Sasha Luccioni, Yacine Jernite, and Emma Strubell. Power Hungry Processing : Watts Driving the Cost of AI Deployment? In The 2024 ACM Conference on Fairness, Accountability, and Transparency, pages 85–99, Rio de Janeiro Brazil, June 2024. ACM. ISBN 9798400704505. doi : 10.1145/3630106.3658542. URL <https://dl.acm.org/doi/10.1145/3630106.3658542>. 18, ii
- David Patterson, Joseph Gonzalez, Urs Holzle, Quoc Le, Chen Liang, Lluís-Miquel Mun-guia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink. Computer, 55(7) : 18–28, July 2022. ISSN 0018-9162, 1558-0814. doi : 10.1109/MC.2022.3148714. URL <https://ieeexplore.ieee.org/document/9810097/>. 18, 43
- Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahñ. Estimation of energy consumption in machine learning. Journal of Parallel and Distributed Computing, 134 :75–88, December 2019. ISSN 07437315. doi : 10.1016/j.jpdc.2019.07.007. URL <https://linkinghub.elsevier.com/retrieve/pii/S0743731518308773>. 20, 23, 34
- Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 6071–6079, Honolulu, HI, July 2017. IEEE. ISBN 978-1-5386-0457-1. doi : 10.1109/CVPR.2017.643. URL <http://ieeexplore.ieee.org/document/8100126/>. 20, 23
- Yunxiang Hu, Yuhao Liu, and Zhuovuan Liu. A Survey on Convolutional Neural Network Accelerators : GPU, FPGA and ASIC. In 2022 14th International Conference on Computer Research and Development (ICCRD), pages 100–107, Shenzhen, China, January 2022. IEEE. ISBN 978-1-72817-721-2. doi : 10.1109/ICCRD54409.2022.9730377. URL <https://ieeexplore.ieee.org/document/9730377/>. 23
- Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. RAPL in Action : Experiences in Using RAPL for Power Measurements. ACM Transactions on Modeling and Performance Evaluation of Computing Systems, 3(2) :

- 1–26, June 2018. ISSN 2376-3639, 2376-3647. doi : 10.1145/3177754. URL <https://dl.acm.org/doi/10.1145/3177754>. 25
- Salah Zidi, Alaeddine Mihoub, Saeed Mian Qaisar, Moez Krichen, and Qasem Abu Al-Haija. Theft detection dataset for benchmarking and machine learning based classification in a smart grid environment. Journal of King Saud University - Computer and Information Sciences, 35(1) :13–25, January 2023. ISSN 13191578. doi : 10.1016/j.jksuci.2022.05.007. URL <https://linkinghub.elsevier.com/retrieve/pii/S1319157822001562>. 26, i
- P.S. Apirajitha. and Anitha Angayarkanni. A design of an adaptive peer-to-peer network to reduce power consumption using cloud computing. In 2012 IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), pages 198–201, Ramanathapuram, India, August 2012. IEEE. ISBN 978-1-4673-2048-1 978-1-4673-2045-0 978-1-4673-2047-4. doi : 10.1109/ICACCCT.2012.6320770. URL <http://ieeexplore.ieee.org/document/6320770/>. 35
- Hagar Hendy and Cory Merkel. Review of spike-based neuromorphic computing for brain-inspired vision : biology, algorithms, and hardware. Journal of Electronic Imaging, 31(01), January 2022. ISSN 1017-9909. doi : 10.1117/1.JEI.31.1.010901. URL <https://www.spiedigitallibrary.org/journals/journal-of-electronic-imaging/volume-31/issue-01/010901/Review-of-spike-based-neuromorphic-computing-for-brain-inspired-vision/10.1117/1.JEI.31.1.010901.full>. 35
- Robert A. Nawrocki, Richard M. Voyles, and Sean E. Shaheen. A Mini Review of Neuromorphic Architectures and Implementations. IEEE Transactions on Electron Devices, 63(10) :3819–3829, October 2016. ISSN 0018-9383, 1557-9646. doi : 10.1109/TED.2016.2598413. URL <http://ieeexplore.ieee.org/document/7549034/>. 35
- Evangelos Stomatias. State-of-the-art deep learning has a carbon emission problem. Can neuromorphic engineering help? Dialogues in Clinical Neuroscience & Mental Health, 3(3) :143–149, May 2020. ISSN 25852795. doi : 10.26386/obrela.v3i3.166. URL <https://doi.org/10.26386/obrela.v3i3.166>. 36
- Javier Del Valle, Juan Gabriel Ramírez, Marcelo J. Rozenberg, and Ivan K. Schuller. Challenges in materials and devices for resistive-switching-based neuromorphic computing. Journal of Applied Physics, 124(21) :211101, December 2018. ISSN 0021-8979, 1089-7550. doi : 10.1063/1.5047800. URL <https://pubs.aip.org/jap/article/124/21/211101/156058/Challenges-in-materials-and-devices-for-resistive>. 36, i
- Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM : Training Multi-Billion Parameter Language Models Using Model Parallelism, 2019. URL <https://arxiv.org/abs/1909.08053>. Version Number : 4. 36
- Tao Luo, Weng-Fai Wong, Rick Siow Mong Goh, Anh Tuan Do, Zhixian Chen, Haizhou Li, Wenyu Jiang, and Weiyun Yau. Achieving Green AI with Energy-Efficient Deep Learning Using Neuromorphic Computing. Communications of the ACM, 66(7) :52–57, July 2023. ISSN 0001-0782, 1557-7317. doi : 10.1145/3588591. URL <https://dl.acm.org/doi/10.1145/3588591>. 37

- Manon Dampfhofer, Thomas Mesquida, Alexandre Valentian, and Lorena Anghel. Are SNNs Really More Energy-Efficient Than ANNs? an In-Depth Hardware-Aware Study. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(3) :731–741, June 2023. ISSN 2471-285X. doi : 10.1109/TETCI.2022.3214509. URL <https://ieeexplore.ieee.org/document/9927729/>. 37, 38, iii
- Fred D. Jordan, Martin Kutter, Jean-Marc Comby, Flora Brozzi, and Ewelina Kurtyś. Open and remotely accessible Neuroplatform for research in wetware computing. *Frontiers in Artificial Intelligence*, 7 :1376042, May 2024. ISSN 2624-8212. doi : 10.3389/frai.2024.1376042. URL <https://www.frontiersin.org/articles/10.3389/frai.2024.1376042/full>. 38
- Lena Smirnova, Brian S. Caffo, David H. Gracias, Qi Huang, Itzy E. Morales Pantoja, Bohao Tang, Donald J. Zack, Cynthia A. Berlinicke, J. Lomax Boyd, Timothy D. Harris, Erik C. Johnson, Brett J. Kagan, Jeffrey Kahn, Alysson R. Muotri, Barton L. Paulhamus, Jens C. Schwamborn, Jesse Plotkin, Alexander S. Szalay, Joshua T. Vogelstein, Paul F. Worley, and Thomas Hartung. Organoid intelligence (OI) : the new frontier in biocomputing and intelligence-in-a-dish. *Frontiers in Science*, 1 : 1017235, February 2023. ISSN 2813-6330. doi : 10.3389/fsci.2023.1017235. URL <https://www.frontiersin.org/articles/10.3389/fsci.2023.1017235/full>. 38
- Julia Lindberg, Yasmine Abdennadher, Jiaqi Chen, Bernard C. Lesieutre, and Line Roald. A Guide to Reducing Carbon Emissions through Data Center Geographical Load Shifting. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, pages 430–436, Virtual Event Italy, June 2021. ACM. ISBN 978-1-4503-8333-2. doi : 10.1145/3447555.3466582. URL <https://dl.acm.org/doi/10.1145/3447555.3466582>. 39, 41
- Gül Nihal Gugul and Sakir Tasdemir. Green IT : An option or necessity. *First international conference on new approaches in engineering proceedings book*, 2022. 40
- G.F. Davies, G.G. Maidment, and R.M. Tozer. Using data centres for combined heating and cooling : An investigation for London. *Applied Thermal Engineering*, 94 :296–304, February 2016. ISSN 13594311. doi : 10.1016/j.applthermaleng.2015.09.111. URL <https://linkinghub.elsevier.com/retrieve/pii/S1359431115010388>. 40
- OpenStudio, France, Arnault Pachot, and Céline Patissier. Towards Sustainable Artificial Intelligence : An Overview of Environmental Protection Uses and Issues. *Green and Low-Carbon Economy*, February 2023. ISSN 29723787. doi : 10.47852/bonviewGLCE3202608. URL <https://ojs.bonviewpress.com/index.php/GLCE/article/view/608>. 45
- Jean-Philippe Nicolai and Lise Peragin. Les certificats de sobriété numérique comme instrument de régulation de la pollution numérique :. *Revue de l'OFCE*, N° 176(1) : 229–249, September 2022. ISSN 1265-9576. doi : 10.3917/reof.176.0229. URL <https://www.cairn.info/revue-de-l-ofce-2022-1-page-229.htm?ref=doi>. 45
- Loïc Lannelongue, Jason Grealey, Alex Bateman, and Michael Inouye. Ten simple rules to make your computing more environmentally sustainable. *PLOS Computational Biology*,

17(9) :e1009324, September 2021. ISSN 1553-7358. doi : 10.1371/journal.pcbi.1009324.
URL <https://dx.plos.org/10.1371/journal.pcbi.1009324>. 47

Annexe 1

Liste des figures

1	Architecture typique d'un <i>data center</i> . Source : PMP, GreenFlex.	4
2	Grandes familles de systèmes de refroidissements (Grégory Lebourg [2024]).	5
3	Architecture minimale d'un ordinateur. Source "Fonctionnement d'un ordinateur".	6
4	Architecture avec une ROM et une RWM. Source : livre Fonctionnement d'un ordinateur.	7
5	Contenu d'un bus informatique. Source : Fonctionnement d'un ordinateur.	9
6	Classement des langages de programmation - juin 2024. Source : TIOBE Softw. BV.	13
7	Représentation générale d'un réseau de neurones artificiels. Source : Lebig-data.	19
8	Fonctionnement de l'algorithme KNN. Source : Zidi et al. [2023].	26
9	Fonctionnement de l'algorithme KMeans. Source : Della Data.	27
10	Base de données MNIST. Source : NIST.	29
11	Résultats de la prédiction du MLP entraîné.	30
12	Résultats de la prédiction du RNN entraîné.	31
13	Résultats de la prédiction du CNN entraîné.	32
14	Comparaison des architectures Von Neumann et des architectures neuromorphiques. Source : Del Valle et al. [2018].	36
15	Comparaison des performances du GPU V100 face à un CPU classique : " <i>A single V100 Tensor Core GPU offers the performance of nearly 32 CPUs</i> ". Source : fiche technique du GPU NVIDIA V100.	37
16	Fonctionnement d'un bioprocasseur. Source : site officiel de FinalSpark. . .	39

17	Intensité carbone des data centers selon leur localisation. Source : Green Algorithms.	41
18	Objectifs ciblés des data centers utilisés afin de définir les leviers d'actions. Source : Schneider Electric.	42

Liste des tableaux

1	Comparaison des valeurs de consommation énergétique en millijoules de différentes architectures informatiques pour un million d'opérations (multiplication de la consommation énergétique typique par opération et du nombre d'opérations).	11
2	Paradigmes de programmation, fonctionnement et exemples de langages. .	12
3	Utilisation de l'outil RAPL pour comparer temps et de énergie consommée par un code de tri par tas et un code de tri par sélection (sur un million de référence) implémentés en C et en Python.	13
4	Apprentissage supervisé (supervised learning).	15
5	Apprentissage non supervisé (unsupervised learning).	15
6	Résultats de la consommation énergétique des algorithmes dans l'étude de Luccioni et al. [2024].	18
7	Les perceptrons multicouches ou MLP (Explications complètes sur : https://neuroconnection.eu/quand-utiliser-les-reseaux-de-neurones-mlp-cnn-et-rnn/).	20
8	Les réseaux de neurones convolutionnels ou CNN (Explications complètes sur : https://fr.blog.businessdecision.com/tutoriel-deep-learning-le-reseau-neuronal-convolutif-cnn/).	21
9	Les réseaux de neurones récurrents ou RNN (Explications complètes sur : https://datavalue-consulting.com/deep-learning-reseaux-neurones-recurrents-rnn/).	21
10	Les réseaux antagonistes génératifs ou GAN (Explications complètes sur : https://khayyam.developpez.com/articles/intelligence-artificielle/tensorflow-gan/).	22

11	Les transformers (utilisés par ex. dans ChatGPT) (Explications complètes sur : https://france.devoteam.com/paroles-dexperts/lstm-transformers-gpt-bert-guide-des-principales-techniques-en-nlp/).	22
12	Résultats des émissions des algorithmes KNN et KMeans.	27
13	Résultats des émissions des algorithmes utilisant Fourier et KMeans.	28
14	Données de performances et de consommation d'énergie obtenues à l'aide de CodeCarbon, suite à l'entraînement du MLP.	29
15	Données de performances et de consommation d'énergie obtenues à l'aide de CodeCarbon, suite à l'entraînement du GRU.	30
16	Données de performances et de consommation d'énergie obtenues à l'aide de CodeCarbon, suite à l'entraînement du CNN.	31
17	Résultats de l'étude Dampfhofer et al. [2023].	38
18	Récapitulatif des leviers d'actions par objectif d'après Grégory Lebourg [2024].	43

Annexe 2

```
1 import ssl
2 import certifi
3 import numpy as np
4 from sklearn.model_selection import train_test_split
5 from sklearn.preprocessing import StandardScaler
6 from sklearn.neighbors import KNeighborsClassifier
7 from sklearn.metrics import accuracy_score
8 from tensorflow.keras.preprocessing.image import load_img,
   img_to_array
9 import matplotlib.pyplot as plt
10 import tensorflow as tf
11 import time # Importation du module time
12 from codecarbon import EmissionsTracker
13
14 ssl._create_default_https_context =
   ssl.create_default_context(cafile=certifi.where())
15
16 # RECOLTE DES DONNEES
17 # chargement des donnees MNIST (ensemble d'entrainement et de
   test)
18 mnist = tf.keras.datasets.mnist
19 (x_train, y_train), (x_test, y_test) = mnist.load_data()
20
21 # Normalisation des donnees
22 x_train = x_train.reshape(-1, 28*28) / 255.0
23 x_test = x_test.reshape(-1, 28*28) / 255.0
24
25 # Diviser les donnees en ensembles de formation et de test
26 x_train, x_val, y_train, y_val = train_test_split(x_train,
   y_train, test_size=0.2, random_state=42)
27
28 # Standardisation des donnees
29 scaler = StandardScaler()
30 x_train = scaler.fit_transform(x_train)
31 x_val = scaler.transform(x_val)
32 x_test = scaler.transform(x_test)
33
34 # Initialisation du tracker CodeCarbon
35 tracker = EmissionsTracker(gpu_ids=[])
36
37 # CREATION DU MODELE KNN
38 knn = KNeighborsClassifier(n_neighbors=3) #on peut ajuster le
   nombre de voisins
39
40 # ENTRAINEMENT DU MODELE
41 print("Debut de l'entrainement du modele KNN")
42 tracker.start() # Demarrage du suivi des emissions
```

```

43 knn.fit(x_train, y_train)
44 emissions_train = tracker.stop() # Arrêt du suivi des missions
45 print(f"Emissions pendant l'entraînement: {emissions_train} kg
    CO2")
46
47 # EVALUATION DU MODELE
48 print("Evaluation du modele sur les donnees de validation")
49 tracker.start() # Demarrage du suivi des emissions pour
    l'evaluation
50 y_val_pred = knn.predict(x_val)
51 val_accuracy = accuracy_score(y_val, y_val_pred)
52 emissions_val = tracker.stop() # Arrêt du suivi des emissions
53 print(f"\nValidation accuracy: {val_accuracy}")
54 print(f"Emissions pendant l'evaluation sur validation:
    {emissions_val} kg CO2")
55
56 print("Evaluation du modele sur les donnees de test")
57 tracker.start() # Demarrage du suivi des emissions pour
    l'evaluation
58 y_test_pred = knn.predict(x_test)
59 test_accuracy = accuracy_score(y_test, y_test_pred)
60 emissions_test = tracker.stop() # Arrêt du suivi des emissions
61 print(f"\nTest accuracy: {test_accuracy}")
62 print(f"Emissions pendant l'evaluation sur test:
    {emissions_test} kg CO2")
63
64 # PREDICTION
65 files = ["zero.png", "one.png", "two.png", "three.png",
    "four.png", "five.png", "six.png", "seven.png"]
66 for file in files:
67     img = load_img("in/" + file, color_mode='grayscale',
        target_size=(28, 28))
68     x = img_to_array(img)
69     x = x.reshape(1, 28*28)
70     x = scaler.transform(x / 255.0)
71
72     # Mesure du temps de pr diction
73     start_time = time.time()
74     y = knn.predict(x)
75     end_time = time.time()
76     elapsed_time = end_time - start_time
77
78     # Prediction de la classe
79     bestClass = y[0]
80     print(f"1/1
        0s {int(elapsed_time * 1000)}ms/step")
81     print(f"{file} => {bestClass}\n")

```


Annexe 3

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from keras.datasets import mnist
4 from sklearn.cluster import MiniBatchKMeans
5 from tensorflow.keras.preprocessing.image import load_img,
    img_to_array
6 from sklearn import metrics
7 import sys
8 import sklearn
9 from codecarbon import EmissionsTracker
10
11 (x_train, y_train), (x_test, y_test) = mnist.load_data()
12
13 # convert each image to 1 dimensional array
14 X = x_train.reshape(len(x_train), -1)
15 Y = y_train
16
17 # normalize the data to 0 - 1
18 X = X.astype(float) / 255.
19
20 print(X.shape)
21 print(X[0].shape)
22
23 n_digits = len(np.unique(y_test))
24 print(n_digits)
25
26 # Initialize KMeans model
27 kmeans = MiniBatchKMeans(n_clusters = n_digits)
28
29 # Fit the model to the training data
30 kmeans.fit(X)
31 kmeans.labels_
32
33
34 def infer_cluster_labels(kmeans, actual_labels):
35     """
36     Associates most probable label with each cluster in KMeans
37     model
38     returns: dictionary of clusters assigned to each label
39     """
40
41     inferred_labels = {}
42
43     for i in range(kmeans.n_clusters):
44
45         # find index of points in cluster
46         labels = []
```

```

46     index = np.where(kmeans.labels_ == i)
47
48     # append actual labels for each point in cluster
49     labels.append(actual_labels[index])
50
51     # determine most common label
52     if len(labels[0]) == 1:
53         counts = np.bincount(labels[0])
54     else:
55         counts = np.bincount(np.squeeze(labels))
56
57     # assign the cluster to a value in the inferred_labels
58     # dictionary
59     if np.argmax(counts) in inferred_labels:
60         # append the new number to the existing array at
61         # this slot
62         inferred_labels[np.argmax(counts)].append(i)
63     else:
64         # create a new array in this slot
65         inferred_labels[np.argmax(counts)] = [i]
66
67     #print(labels)
68     #print('Cluster: {}, label: {}'.format(i,
69         np.argmax(counts)))
70
71     return inferred_labels
72
73 def infer_data_labels(X_labels, cluster_labels):
74     """
75     Determines label for each array, depending on the cluster it
76     has been assigned to.
77     returns: predicted labels for each array
78     """
79
80     # empty array of len(X)
81     predicted_labels = np.zeros(len(X_labels)).astype(np.uint8)
82
83     for i, cluster in enumerate(X_labels):
84         for key, value in cluster_labels.items():
85             if cluster in value:
86                 predicted_labels[i] = key
87
88     return predicted_labels
89
90 # test the infer_cluster_labels() and infer_data_labels()
91 # functions
92 cluster_labels = infer_cluster_labels(kmeans, Y)
93 X_clusters = kmeans.predict(X)

```

```

90 predicted_labels = infer_data_labels(X_clusters, cluster_labels)
91 print (predicted_labels[:20])
92 print (Y[:20])
93
94
95 # Calculate and print metrics
96 def calculate_metrics(estimator, data, labels):
97     print('Number of Clusters: {}'.format(estimator.n_clusters))
98     print('Inertia: {}'.format(estimator.inertia_))
99     print('Homogeneity:
        {}'.format(metrics.homogeneity_score(labels,
        estimator.labels_)))
100
101 clusters = [10, 16, 36, 64, 144, 256]
102
103 # test different numbers of clusters
104 for n_clusters in clusters:
105     tracker = EmissionsTracker()
106     tracker.start()
107     estimator = MiniBatchKMeans(n_clusters = n_clusters)
108     estimator.fit(X)
109
110     # print cluster metrics
111     calculate_metrics(estimator, X, Y)
112
113     # determine predicted labels
114     cluster_labels = infer_cluster_labels(estimator, Y)
115     predicted_Y = infer_data_labels(estimator.labels_,
        cluster_labels)
116
117     # calculate and print accuracy
118     print('Accuracy: {}\n'.format(metrics.accuracy_score(Y,
        predicted_Y)))
119     emissions = tracker.stop()
120     print(f"Emissions : {emissions} kg CO2")
121
122 # test kmeans algorithm on testing dataset
123 X_test = x_test.reshape(len(x_test),-1)
124 X_test = X_test.astype(float) / 255.
125
126 tracker = EmissionsTracker()
127 tracker.start()
128 # initialize and fit KMeans algorithm on training data
129 kmeans = MiniBatchKMeans(n_clusters = 256)
130 kmeans.fit(X)
131 cluster_labels = infer_cluster_labels(kmeans, Y)
132 emissions = tracker.stop()
133 print(f"Emissions : {emissions} kg CO2")
134

```

```
135 # predict labels for testing data
136 test_clusters = kmeans.predict(X_test)
137 print(type(X_test))
138 predicted_labels = infer_data_labels(kmeans.predict(X_test),
    cluster_labels)
139
140 # calculate and print accuracy
141 print('Accuracy: {}'.format(metrics.accuracy_score(y_test,
    predicted_labels)))
142
143 #Test sur nos images
144 files = ["zero.png", "one.png", "two.png", "three.png",
    "four.png", "five.png", "six.png", "seven.png"]
145
146 for file in files:
147     img = load_img("in/" + file, color_mode='grayscale',
        target_size=(28, 28))
148     x = img_to_array(img).reshape(1, -1).astype(float) / 255.0
149     cluster = kmeans.predict(x)
150     predicted_label = infer_data_labels(kmeans.predict(x),
        cluster_labels)
151     print(file + " => " + str(predicted_label) + "\n")
```

Annexe 4

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from codecarbon import EmissionsTracker
4 import time
5
6 # Dimensions de l'image
7 rows, cols = 256, 256
8
9 # Creation de l'image de reference avec des cercles
10 image_ref = np.zeros((rows, cols))
11
12 # Dessiner quelques cercles
13 for x in range(rows):
14     for y in range(cols):
15         if (x - 128)**2 + (y - 128)**2 < 30**2 or (x - 64)**2 +
            (y - 64)**2 < 20**2 or (x - 192)**2 + (y - 192)**2 <
            20**2:
16             image_ref[x, y] = 1
17
18 # Affichage de l'image de reference
19 plt.imshow(image_ref, cmap='gray')
20 plt.title('Image de r e f e r e n c e')
21 plt.axis('off')
22 plt.show()
23
24 tracker = EmissionsTracker()
25 tracker.start()
26
27 # Calcul de la transformee de Fourier de l'image de reference
28 frequencies = np.fft.fft2(image_ref)
29
30 # Inverse de la transformee de Fourier pour obtenir l'image
31 start_time = time.time()
32 image_fft = np.fft.ifft2(frequencies).real
33 end_time = time.time()
34
35 generation_time = end_time - start_time
36
37 emissions = tracker.stop()
38
39 # Affichage de l'image generee avec la Transformee de Fourier
40 plt.imshow(image_fft, cmap='gray')
41 plt.title('Image generee avec la Transformee de Fourier')
42 plt.axis('off')
43 plt.show()
44
45 print(f"Temps de generation pour la Transformee de Fourier:
```

```
    {generation_time} secondes")  
46 print(f"Emissions for Fourier transform: {emissions} kg CO2")
```

Annexe 5

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from codecarbon import EmissionsTracker
4 import time
5
6 # Dimensions de l'image
7 rows, cols = 256, 256
8
9 # Cr ation de l'image de reference avec des cercles
10 image_ref = np.zeros((rows, cols))
11
12 # Dessiner quelques cercles
13 for x in range(rows):
14     for y in range(cols):
15         if (x - 128)**2 + (y - 128)**2 < 30**2 or (x - 64)**2 +
            (y - 64)**2 < 20**2 or (x - 192)**2 + (y - 192)**2 <
            20**2:
16             image_ref[x, y] = 1
17
18 # Affichage de l'image de reference
19 plt.imshow(image_ref, cmap='gray')
20 plt.title('Image de r f rence')
21 plt.axis('off')
22 plt.show()
23
24 from sklearn.cluster import KMeans
25
26 tracker = EmissionsTracker()
27 tracker.start()
28
29 # Redimensionner l'image de reference pour K-Means
30 X = image_ref.reshape(-1, 1)
31
32 # Appliquer K-Means pour regrouper les pixels en 2 clusters
33 start_time = time.time()
34 kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
35 end_time = time.time()
36
37 generation_time = end_time - start_time
38
39 # Attribuer des couleurs aux clusters
40 image_kmeans = kmeans.labels_.reshape((rows, cols))
41
42 emissions = tracker.stop()
43
44 # Affichage de l'image generee avec K-Means
45 plt.imshow(image_kmeans, cmap='gray')
```



```
46 plt.title('Image genereee avec K-Means')
47 plt.axis('off')
48 plt.show()
49
50 print(f"Temps de generation pour K-Means: {generation_time}
      secondes")
51 print(f"Emissions for K-Means: {emissions} kg CO2")
```

Annexe 6

```

1 #INSTALLATION DES BIBLIOTHEQUES
2 import tensorflow as tf
3 print(tf.__version__)
4 from tensorflow.keras.preprocessing.image import load_img,
    img_to_array
5 import matplotlib.pyplot as plt
6 from sklearn.metrics import roc_curve, auc
7 import numpy as np
8
9 #RECOLTE DES DONNEES
10 #chargement des donnees MNIST (ensemble d'entrainement et de
    test)
11 mnist = tf.keras.datasets.mnist
12 (x_train, y_train), (x_test, y_test) = mnist.load_data() #image
    en entr e (x) et classe associ e en sortie (y)
13
14 # Normalisation des donnees
15 x_train, x_test = x_train / 255.0, x_test / 255.0
16
17 #CREATION DU RESEAU DE NEURONES
18 model = tf.keras.models.Sequential([
19     # premiere couche pour recevoir tous les pixels en entree :
20     tf.keras.layers.Flatten(input_shape=(28, 28)),
21     #couches cachees : ATTENTION TROP DE COUCHES CACHEES PEUT
        MENER AU SURAPPRENTISSAGE
22     tf.keras.layers.Dense(128, activation='relu'), #tanh et
        sigmoid moins bien que relu quand on teste
23     tf.keras.layers.Dense(128, activation='relu'),
24     tf.keras.layers.Dense(128, activation='relu'),
25     tf.keras.layers.Dense(128, activation='relu'),
26     tf.keras.layers.Dense(128, activation='relu'),
27     tf.keras.layers.Dense(128, activation='relu'),
28     tf.keras.layers.Dense(128, activation='relu'),
29     tf.keras.layers.Dense(128, activation='relu'),
30     tf.keras.layers.Dense(128, activation='relu'),
31     tf.keras.layers.Dense(128, activation='relu'),
32     # derni re couche de 10 neurones, enti rement connectee
        la couche pr c dente
33     tf.keras.layers.Dense(10),
34     # traitement pour ramener toutes les valeurs entre 0 et 1
35     tf.keras.layers.Softmax()
36 ])
37
38 #ENTRAINEMENT DU MODELE SUR LES DONNEES D'ENTRAINEMENT
39 # Compilation du modele avec une fonction de perte, un
    optimiseur et une metrique
40 model.compile(optimizer='adam',

```

```
41         loss='sparse_categorical_crossentropy',
42         metrics=['accuracy'])
43
44 history=model.fit(x_train, y_train, epochs=5) #epochs = nb
         d'iterations
45
46 #EVALUATION DU MODELE SUR LES DONNEES DE TEST
47 test_loss, test_acc = model.evaluate(x_test, y_test)
48 print('\nTest accuracy:', test_acc)
49
50 #PREDICTION
51 #Chargement et pre-traitement
52 files = ["zero.png", "one.png", "two.png", "three.png",
         "four.png", "five.png", "six.png", "seven.png"]
53 for file in files:
54     img = load_img("in/" + file, color_mode='grayscale')
55     x = img_to_array(img)
56     x = x.reshape(1,28,28)
57     x = x / 255.0
58     #Prediction de la classe
59     y = model.predict(x)
60     y = y[0]
61     bestClass = y.argmax()
62     print(file + " => " + str(bestClass) + "\n")
```

Annexe 7

```

1 # INSTALLATION DES BIBLIOTHEQUES
2 import tensorflow as tf
3 print(tf.__version__)
4 from tensorflow.keras.preprocessing.image import load_img,
    img_to_array
5 import matplotlib.pyplot as plt
6 import numpy as np
7 from codecarbon import EmissionsTracker
8
9 # RECOLTE DES DONNEES
10 # chargement des donnees MNIST (ensemble d'entrainement et de
    test)
11 mnist = tf.keras.datasets.mnist
12 (x_train, y_train), (x_test, y_test) = mnist.load_data() # image
    en entree (x) et classe associee en sortie (y)
13
14 # Normalisation des donnees
15 x_train, x_test = x_train / 255.0, x_test / 255.0
16
17 # Reshape des donnees pour GRU
18 x_train = x_train.reshape(-1, 28, 28) # GRU attend un input de
    forme (batch_size, time_steps, input_dim)
19 x_test = x_test.reshape(-1, 28, 28)
20
21 # CREATION DU RESEAU GRU
22 model = tf.keras.models.Sequential([
23     # Couche GRU : 2 couches cachees
24     tf.keras.layers.GRU(128, input_shape=(28, 28),
        return_sequences=True),
25     tf.keras.layers.GRU(128),
26     # derniere couche de 10 neurones, entierement connectee
        la couche precedente
27     tf.keras.layers.Dense(10, activation='softmax')
28 ])
29
30 # ENTRAINEMENT DU MODELE SUR LES DONNEES D'ENTRAINEMENT
31 # Compilation du modele avec une fonction de perte, un
    optimiseur et une metrique
32 model.compile(optimizer='adam',
33               loss='sparse_categorical_crossentropy',
34               metrics=['accuracy'])
35
36 # Suivi des emissions
37 tracker = EmissionsTracker()
38 tracker.start()
39
40 # Entrainement du modele sur les donnees d'entrainement

```

```
41 print("D but de l'entra nement")
42 history = model.fit(x_train, y_train, epochs=5) # epochs = nb
         d'iterations
43 print("Fin de l'entra nement")
44
45 # Fin du suivi des emissions
46 emissions = tracker.stop()
47 print(f"Emissions: {emissions} kg CO2")
48
49 # EVALUATION DU MODELE SUR LES DONNEES DE TEST
50 test_loss, test_acc = model.evaluate(x_test, y_test)
51 print('\nTest accuracy:', test_acc)
52
53 # PREDICTION
54 # Chargement et pre-traitement
55 files = ["zero.png", "one.png", "two.png", "three.png",
         "four.png", "five.png", "six.png", "seven.png"]
56 for file in files:
57     img = load_img("in/" + file, color_mode='grayscale',
         target_size=(28, 28))
58     x = img_to_array(img)
59     x = x.reshape(1, 28, 28)
60     x = x / 255.0
61
62     y = model.predict(x)
63     y = y[0]
64     bestClass = y.argmax()
65     print(file + " => " + str(bestClass) + "\n")
```

Annexe 8

```

1 #INSTALLATION DES BIBLIOTHEQUES
2 import tensorflow as tf
3 print(tf.__version__)
4 from tensorflow.keras.preprocessing.image import load_img,
    img_to_array
5 import matplotlib.pyplot as plt
6 from sklearn.metrics import roc_curve, auc
7 import numpy as np
8
9 # RECOLTE DES DONNEES
10 # Chargement des donnees MNIST (ensemble d'entrainement et de
    test)
11 mnist = tf.keras.datasets.mnist
12 (x_train, y_train), (x_test, y_test) = mnist.load_data() #
    image en entree (x) et classe associee en sortie (y)
13
14 # Normalisation des donnees
15 x_train, x_test = x_train / 255.0, x_test / 255.0
16
17 # "Reshape" des donnees pour les rendre compatibles avec les CNNs
18 x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
19 x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
20
21 # CREATION DU RESEAU DE NEURONES CONVOLUTIONNEL
22 model = tf.keras.models.Sequential([
23     # Premiere couche de convolution
24     tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
        input_shape=(28, 28, 1)), #fonction d'activation choisie :
        relu
25     tf.keras.layers.MaxPooling2D((2, 2)),
26
27     # Deuxieme couche de convolution
28     tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
29     tf.keras.layers.MaxPooling2D((2, 2)),
30
31     # Troisieme couche de convolution
32     tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
33
34     # Couche Flatten pour convertir les matrices 2D en vecteurs
        1D
35     tf.keras.layers.Flatten(),
36
37     # Couches entierement connectees
38     tf.keras.layers.Dense(128, activation='relu'),
39     tf.keras.layers.Dense(10, activation='softmax') # Derniere
        couche avec softmax pour la classification des 10 classes
40 ])

```

```
41
42 # ENTRAÎNEMENT DU MODÈLE SUR LES DONNÉES D'ENTRAÎNEMENT
43 # Compilation du modèle avec une fonction de perte, un
    optimiseur et une métrique
44 model.compile(optimizer='adam',
45               loss='sparse_categorical_crossentropy',
46               metrics=['accuracy'])
47
48 history = model.fit(x_train, y_train, epochs=5) # epochs = nb
    d'iterations
49
50 # ÉVALUATION DU MODÈLE SUR LES DONNÉES DE TEST
51 test_loss, test_acc = model.evaluate(x_test, y_test)
52 print('\nTest accuracy:', test_acc)
53
54 # PREDICTION
55 # Chargement et pré-traitement des nouvelles images
56 files = ["zero.png", "one.png", "two.png", "three.png",
    "four.png", "five.png", "six.png", "seven.png"]
57 for file in files:
58     img = load_img("in/" + file, color_mode='grayscale')
59     x = img_to_array(img)
60     x = x.reshape(1, 28, 28, 1)
61     x = x / 255.0
62     # Prédiction de la classe
63     y = model.predict(x)
64     y = y[0]
65     bestClass = y.argmax()
66     print(file + " => " + str(bestClass) + "\n")
```

Annexe 9

```

1 #ATTENTION
2 #Gourmand en ressources
3 #-> ne peut pas etre simplement lance sur un ordinateur portable
4
5 #INSTALLATION DES BIBLIOTHEQUES
6 import glob
7 import imageio
8 import matplotlib.pyplot as plt
9 import numpy as np
10 import os
11 import PIL
12 import tensorflow as tf
13 tf.__version__
14 from tensorflow.keras import layers
15 from codecarbon import EmissionsTracker
16 import time
17 from IPython import display
18
19 #RECOLTE DES DONNEES
20 (train_images, train_labels), (_, _) =
    tf.keras.datasets.mnist.load_data()
21 train_images = train_images.reshape(train_images.shape[0], 28,
    28, 1).astype('float32')
22 train_images = (train_images - 127.5) / 127.5 # Normalize the
    images to [-1, 1]
23 BUFFER_SIZE = 60000
24 BATCH_SIZE = 256
25 # Batch and shuffle the data
26 train_dataset =
    tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(
27
28
29 #DEFINITION DES FONCTIONS UTILES (GENERATEUR, DISCRIMINATEUR,
    ERREUR_GEN, ERREUR_DIS)
30 def make_generator_model():
31     model = tf.keras.Sequential()
32     model.add(layers.Dense(7*7*256, use_bias=False,
        input_shape=(100,)))
33     model.add(layers.BatchNormalization())
34     model.add(layers.LeakyReLU())
35
36     model.add(layers.Reshape((7, 7, 256)))
37     assert model.output_shape == (None, 7, 7, 256) # Note: None
        is the batch size
38
39     model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1,
        1), padding='same', use_bias=False))

```



```

40     assert model.output_shape == (None, 7, 7, 128)
41     model.add(layers.BatchNormalization())
42     model.add(layers.LeakyReLU())
43
44     model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2),
45         padding='same', use_bias=False))
46     assert model.output_shape == (None, 14, 14, 64)
47     model.add(layers.BatchNormalization())
48     model.add(layers.LeakyReLU())
49
50     model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2),
51         padding='same', use_bias=False, activation='tanh'))
52     assert model.output_shape == (None, 28, 28, 1)
53
54     return model
55 generator = make_generator_model()
56
57 noise = tf.random.normal([1, 100])
58 generated_image = generator(noise, training=False)
59
60 #plt.imshow(generated_image[0, :, :, 0], cmap='gray')
61
62 def make_discriminator_model():
63     model = tf.keras.Sequential()
64     model.add(layers.Conv2D(64, (5, 5), strides=(2, 2),
65         padding='same',
66         input_shape=[28, 28, 1]))
67     model.add(layers.LeakyReLU())
68     model.add(layers.Dropout(0.3))
69
70     model.add(layers.Conv2D(128, (5, 5), strides=(2, 2),
71         padding='same'))
72     model.add(layers.LeakyReLU())
73     model.add(layers.Dropout(0.3))
74
75     model.add(layers.Flatten())
76     model.add(layers.Dense(1))
77
78     return model
79
80 discriminator = make_discriminator_model()
81 decision = discriminator(generated_image)
82 print (decision)
83
84 # Cette methode renvoie une fonction d'aide pour calculer la
85     perte d'entropie croisee
86 cross_entropy =
87     tf.keras.losses.BinaryCrossentropy(from_logits=True)

```

```

83 def discriminator_loss(real_output, fake_output):
84     real_loss = cross_entropy(tf.ones_like(real_output),
85                               real_output)
86     fake_loss = cross_entropy(tf.zeros_like(fake_output),
87                               fake_output)
88     total_loss = real_loss + fake_loss
89     return total_loss
90
91 def generator_loss(fake_output):
92     return cross_entropy(tf.ones_like(fake_output), fake_output)
93
94 generator_optimizer = tf.keras.optimizers.Adam(1e-4)
95 discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
96
97 checkpoint_dir = './training_checkpoints'
98 checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
99 checkpoint =
100     tf.train.Checkpoint(generator_optimizer=generator_optimizer,
101                          discriminator_optimizer=discriminator_optimizer,
102                          generator=generator,
103                          discriminator=discriminator)
104
105 EPOCHS = 5
106 noise_dim = 100
107 num_examples_to_generate = 16
108
109 # pour visualiser les progres dans le GIF anime
110 seed = tf.random.normal([num_examples_to_generate, noise_dim])
111
112 @tf.function
113 def train_step(images):
114     noise = tf.random.normal([BATCH_SIZE, noise_dim])
115
116     with tf.GradientTape() as gen_tape, tf.GradientTape() as
117         disc_tape:
118         generated_images = generator(noise, training=True)
119
120         real_output = discriminator(images, training=True)
121         fake_output = discriminator(generated_images,
122                                   training=True)
123
124         gen_loss = generator_loss(fake_output)
125         disc_loss = discriminator_loss(real_output, fake_output)
126
127     gradients_of_generator = gen_tape.gradient(gen_loss,
128                                                generator.trainable_variables)
129     gradients_of_discriminator = disc_tape.gradient(disc_loss,

```

```

        discriminator.trainable_variables)
126
127     generator_optimizer.apply_gradients(zip(gradients_of_generator,
        generator.trainable_variables))
128     discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
        discriminator.trainable_variables))
129
130 def train(dataset, epochs):
131     for epoch in range(epochs):
132         start = time.time()
133
134         for image_batch in dataset:
135             train_step(image_batch)
136
137         # Production des images pour le GIF
138         display.clear_output(wait=True)
139         generate_and_save_images(generator,
140                                 epoch + 1,
141                                 seed)
142
143         # Enregistrement du modele tous les 15 epochs
144         if (epoch + 1) % 15 == 0:
145             checkpoint.save(file_prefix = checkpoint_prefix)
146
147         print ('Time for epoch {} is {} sec'.format(epoch + 1,
        time.time()-start))
148
149         # Generer apres la derniere epoque
150         display.clear_output(wait=True)
151         generate_and_save_images(generator,
152                                 epochs,
153                                 seed)
154
155 def generate_and_save_images(model, epoch, test_input):
156     predictions = model(test_input, training=False)
157
158     fig = plt.figure(figsize=(4, 4))
159
160     for i in range(predictions.shape[0]):
161         plt.subplot(4, 4, i+1)
162         plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5,
        cmap='gray')
163         plt.axis('off')
164
165     plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
166     plt.show()
167
168 # Utilisation de codecarbon
169 tracker = EmissionsTracker()

```

```
170 tracker.start()
171
172 train(train_dataset, EPOCHS)
173
174 emissions = tracker.stop()
175 print(f"Emissions: {emissions} kg CO2")
176
177 checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
178
179 # Affichage d'une seule image en utilisant le numero d'epoch
180 def display_image(epoch_no):
181     return
182     PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))
183 display_image(EPOCHS)
184
185 anim_file = 'dcgan.gif'
186
187 with imageio.get_writer(anim_file, mode='I') as writer:
188     filenames = glob.glob('image*.png')
189     filenames = sorted(filenames)
190     for filename in filenames:
191         image = imageio.imread(filename)
192         writer.append_data(image)
193     image = imageio.imread(filename)
194     writer.append_data(image)
195
196 import tensorflow_docs.vis.embed as embed
197 embed.embed_file(anim_file)
```

★ ★ ★