Département de génie informatique et de génie logiciel

INF8970

Projet final en génie informatique

Rapport final

1chipML

Équipe No. 7

Nicholas Legrand

Gaya Mehenni

Félix-Antoine Constantin

Alexis Foulon

Paul-André Bisson

Florence Cloutier

Avril 2023

Table des matières

1. Changement dans le contexte du projet

1.1 Description des changements imposés par le client depuis la mi-session

Suite à la présentation de mi-session, de nombreux changements ont été proposés par le client pour garantir le bon déroulement du projet jusqu'à la fin de la session. Ces changements permettent majoritairement de mieux définir les critères du client pour ainsi s'assurer que le produit final corresponde à ses attentes.

Premièrement, à la demande du client, une version spécialisée des fonctions trigonométriques sinus et cosinus a été développée. En effet, l'implémentation proposée par la librairie standard n'est pas efficace sur les microcontrôleurs. Ainsi, l'appel à ces méthodes présente un coût très élevé pour les algorithmes qui font grandement usage de celles-ci. Une approximation de ces méthodes a donc été développée pour notre librairie. Cette approximation permet d'obtenir les valeurs désirées jusqu'à huit fois plus rapidement selon de la précision nécessaire.

Deuxièmement, un pipeline de validation a été ajouté au projet. Celui-ci a pour but de vérifier automatiquement, lors des revues de code, que les développements qui sont faits respectent les critères imposés par le projet. En effet, le standard proposé par le projet LLVM [1] doit être respecté lors de l'ajout de nouveaux algorithmes. Ce standard peut être facilement validé automatiquement et il n'est pas nécessaire d'avoir des personnes attribuées à la validation lors de la revue de code. Un second pipeline a aussi été développé pour compiler automatiquement la documentation. Celui-ci est utilisé pour s'assurer que la documentation présente sur le dépôt est mise à jour automatiquement lors de l'ajout de nouveaux algorithmes. Ce pipeline n'a cependant pas encore été intégré puisque des modifications ne pouvant seulement être effectuées par la fondation Linux sont requises dans les paramètres du dépôt.

Troisièmement, des validations ont été faites par le client sur la documentation produite à ce jour. Pour s'assurer que le projet est bien documenté et simple à utiliser pour les utilisateurs externes, le client a relu la documentation produite pour les algorithmes développés avant la mi-session. Suite à cette relecture, le client a apporté des commentaires pertinents permettant d'améliorer la documentation disponible et de permettre une meilleure compréhension des algorithmes disponibles pour les nouveaux utilisateurs.

Quatrièmement, le client a proposé plusieurs prototypes à implémenter pour les algorithmes devant être développés suite à la mi-session. En effet, notre équipe savait que des prototypes allaient devoir être développés puisque cela faisait partie des besoins du client, mais celui-ci n'avait pas indiqué exactement quels prototypes allaient devoir être développés. Ainsi, le client a proposé d'implémenter, pour l'algorithme de décomposition LU et de différences finies, un système de recommandation. Il a aussi proposé d'implémenter, pour les divers algorithmes statistiques, un système de regroupement utilisant l'algorithme de K-Moyennes.

Finalement, suite à l'ajout de ces nouveaux requis, le client a pris la décision de retirer l'algorithme du bandit manchot. En effet, cet algorithme avait été analysé par notre équipe au début de la session pour choisir les méthodes à implémenter et représentait une charge de travail considérable. Le client a donc préféré le retirer pour permettre à l'équipe de peaufiner les algorithmes déjà implémentés et donc d'optimiser ceux-ci pour qu'ils soient plus simples à utiliser ou encore s'exécutent plus rapidement.

1.2 Difficultés techniques nouvellement apparues

La première difficulté technique nouvellement apparue est reliée au code qui était déjà présent avant notre entrée dans le projet. En effet, lors du développement, nous avons remarqué qu'une partie du code de base n'avait pas le comportement attendu. Puisque cette partie était essentielle à la réalisation de plusieurs de nos modules, nous nous sommes alloué du temps afin de régler cette dernière.

Une autre difficulté technique nouvellement apparue est liée à l'optimisation de la mémoire pour certains algorithmes, tels que l'algorithme génétique et l'algorithme de Monte Carlo pour l'apprentissage par renforcement. Étant donné la nature de ces algorithmes, il est nécessaire de conserver en mémoire différentes données afin d'assurer l'obtention des meilleurs résultats. Après avoir placé ces algorithmes sur des microcontrôleurs, l'équipe a rapidement réalisé que ces derniers devaient être modifiés pour permettre leur utilisation optimale sur des microcontrôleurs, sans que des problèmes surgissent. Il a alors fallu opter pour des implémentations plus lentes, mais qui permettent d'économiser plus de mémoire. En ce qui concerne l'optimisation de la vitesse des algorithmes, plusieurs recherches ont été faites afin de trouver des façons de les accélérer. La difficulté principale associée aux optimisations était de trouver des améliorations qui s'appliquent aux microcontrôleurs. Plusieurs options trouvées initialement fonctionnent très bien sur ordinateur, mais ont peu d'impact une fois placées sur des microcontrôleurs. Il a alors fallu adapter les solutions trouvées pour qu'elles soient efficaces selon le contexte de développement.

2. Description de l'architecture finale du système (Q5.2 – Q5.3 – Q5.4)

2.1 Résumé de ce qu'était le système à la mi-session et ce qu'il est maintenant (Q4.5 et Q4.6)

2.1.1 Système général

Le système général est une librairie à code source ouvert, développée entièrement en C, qui n'utilise aucune librairie externe. Cette librairie contient des algorithmes fréquemment utilisés dans le domaine de l'apprentissage machine et de l'intelligence artificielle. Dans la librairie, un fichier d'en-tête est chargé d'inclure tous les algorithmes développés, afin de faciliter l'inclusion de ceux-ci. De plus, chaque algorithme possède son propre en-tête pour permettre au client et aux futurs utilisateurs d'inclure

uniquement les algorithmes dont ils ont besoin. Également, les algorithmes qui utilisent les mêmes fonctions utilitaires dépendent des mêmes fichiers utilitaires.

À la mi-session, 6 algorithmes étaient partiellement développés dans la librairie. Chaque algorithme avait un test de base pour s'assurer de son bon fonctionnement et 4 parmi les 6 algorithmes possédaient également un prototype qui s'exécutait sur un Arduino, démontrant un cas d'usage de l'algorithme. La librairie comprend maintenant 9 algorithmes développés ayant chacun un test de base ainsi qu'un prototype. Ces 9 algorithmes sont détaillés dans les sections 2.1.2 à 2.1.11. Un autre algorithme de calcul rapide des opérations sinus et cosinus a également été implémenté dans la deuxième moitié de la session. De plus, dans le dépôt Github, un pipeline a été ajouté depuis la mi-session pour s'assurer du bon fonctionnement de nouveau code qui est intégré dans la branche principale du dépôt.

2.1.2 Transformée de Fourier rapide (FFT) et transformée de Fourier discrète (DFT)

L'algorithme de la FFT est généralement utilisé pour transformer un signal d'un certain domaine, comme le temps, vers un domaine de fréquences. Il existe plusieurs solutions possibles en ce qui concerne l'implémentation des algorithmes FFT et DFT. Pour la DFT et la FFT, il existe de nombreuses techniques visant à accélérer les différents calculs qui ont lieu. La solution initiale d'opter pour un algorithme itératif [2] dans la conception de la FFT a été conservée, car il a été observé que cette méthode conserve l'efficacité des autres méthodes possibles, tout en permettant de satisfaire avec facilité les différentes contraintes matérielles. Également, la décision de ne pas précalculer les facteurs multiplicatifs a été conservée, car il n'a pas été possible d'observer les avantages de cette méthode selon le contexte de développement.

2.1.3 Lanczos

L'algorithme de Lanczos est utilisé pour calculer les vecteurs propres et valeurs propres associées à une matrice symétrique autoadjointe. Cet algorithme prend en entrée une matrice et retourne deux matrices en sortie. Ces matrices ne contiennent pas directement les vecteurs propres et valeurs propres, mais fournissent plutôt des matrices dont ces valeurs sont plus simples à calculer. Plus exactement, l'algorithme retourne une matrice tridiagonale qui permet de plus facilement calculer les valeurs propres et une seconde matrice utilisée pour transformer les vecteurs propres vers l'espace initial.

L'algorithme est assez simple à implémenter et peu de variantes existent à son sujet. En effet, le seul défi que cet algorithme pose est l'instabilité numérique produite par son exécution. De nombreuses méthodes existent pour pallier ce problème. L'ordre des opérations peut notamment être modifié. C. Paige [3] a démontré que l'ordre des opérations avait un impact sur l'instabilité numérique produite par l'algorithme. Une attention particulière a donc été portée à celle-ci. De plus, de nombreuses méthodes existent pour tenter de garder la matrice de vecteurs propre orthonormée et donc d'éviter de perdre certaines caractéristiques importantes de l'algorithme. Par exemple,

une réorthogonalisation peut être effectuée sur la matrice. Rien n'a été modifié par rapport à la mi-session.

2.1.4 Jacobi

L'algorithme de Jacobi est un algorithme de résolution de valeurs propres et de vecteurs propres. En effet, étant donné une matrice symétrique, l'algorithme est capable de calculer les vecteurs et valeurs propres de la matrice.

Plusieurs implémentations de cet algorithme existent, chacune avec ses propres avantages. Par exemple, la méthode de Jacobi cyclique se retrouve à être plus efficace dans le cas de certaines matrices et la méthode classique l'est dans d'autres cas. La différence principale entre ces deux méthodes réside dans le choix de l'élément à éliminer lors d'une itération de l'algorithme. Puis, des implémentations existent pour être plus efficaces en moyenne dans le temps. Cependant, ces dernières utilisent plus d'espace et sont parfois sous la protection de droits d'auteurs. Le fait d'implémenter l'algorithme sur un microcontrôleur implique une bonne gestion de la mémoire et de l'efficacité de l'algorithme. Notamment, la version présentée en Fortran dans le livre Numerical Recipes [2] semble trouver un juste milieu entre simplicité d'implémentation, efficacité et utilisation de la mémoire.

2.1.5 Descente du gradient

L'algorithme de la descente du gradient conjuguée utilisé se base sur les équations trouvées dans le livre Numerical Recipes [2].

Cet algorithme tient compte de deux vecteurs. Le gradient de la fonction au point que nous observons, soit le vecteur g. Le vecteur conjugué pour le point que nous observons, soit le vecteur h. L'algorithme consiste à obtenir le vecteur conjugué à partir du gradient à chaque itération. Le vecteur conjugué s'obtient grâce au calcul d'un facteur d'ajustement $\boldsymbol{\gamma}$ que l'on applique au vecteur g*. Par la suite, une technique de recherche linéaire est utilisée afin de trouver le minimum de la fonction dans la direction donnée par le vecteur conjugué. On se déplace ensuite à ce point et on recommence l'algorithme jusqu'à ce que le déplacement vers le nouveau point soit inférieur à la tolérance spécifiée par l'utilisateur, ce qui indique que le minimum a été trouvé.

La méthode de recherche linéaire encore utilisée depuis la mi-session est la méthode de Brent [4] décrite dans le livre Numerical Recipes [2]. Celle-ci utilise un mélange d'interpolation quadratique et de recherche par bissection afin de trouver le minimum de la fonction dans la direction du conjugué.

2.1.6 Algorithme génétique

L'algorithme génétique est une métaheuristique utilisée pour des problèmes d'optimisation globaux, c'est-à-dire qu'il permet d'optimiser des fonctions qui possèdent plusieurs optimums locaux, ce qui est sa plus grande force face à un algorithme d'escalade traditionnel. Pour ce faire, l'algorithme simule le comportement de l'évolution

d'une population. Initialement, une population est créée de manière aléatoire. Ensuite, au travers de diverses itérations, une série d'opérations génétiques sont appliquées de manière à optimiser les résultats obtenus. Les opérations appliquées sur la population sont habituellement le croisement et la mutation. Le croisement est utilisé pour créer des enfants à partir de deux membres de la population qui sont choisis en fonction de leur force. La force d'une solution est évaluée à l'aide d'une fonction objective. Les nouveaux enfants ont ensuite la possibilité de subir une mutation aléatoire de l'un de leurs gènes. Lorsque suffisamment d'enfants sont créés, la génération précédente est remplacée. Le processus se répète jusqu'à ce qu'une solution acceptable soit produite ou lorsque le nombre d'itérations maximum est atteint [5].

Cet algorithme peut toutefois consommer énormément de mémoire pour l'enregistrement de la population , la génération des enfants et des forces de chaque membre de la population. Une deuxième limite de cet algorithme est qu'il peut trouver rapidement une bonne solution, mais il peut prendre beaucoup d'itérations pour trouver l'optimum global [5].

Plusieurs variantes de cet algorithme existent. Une des différences clés de ces variantes est le processus de sélection des parents. Une des méthodes de choix les plus populaires est la sélection des parents proportionnés, c'est-à-dire que les parents avec la meilleure aptitude vont davantage être sélectionnés. Une deuxième méthode de sélection qui existe est la sélection par tournoi. L'algorithme choisit aléatoirement un certain nombre de membres de la population et retourne le meilleur membre. La dernière méthode de sélection est la sélection par rang. Cette variante est légèrement différente de la méthode de sélection proportionnée. Elle classe les solutions selon leur aptitude et choisit ensuite les parents de manière aléatoire. Les chances qu'une solution soit choisie augmentent avec son rang. Une autre variante populaire de cet algorithme implémente l'élitisme, c'est-à-dire de directement passer les meilleures solutions aux prochaines générations. [5]

Le croisement des parents peut également être fait de plusieurs façons. L'utilisation d'un index aléatoire pour déterminer le point de croisement est une des méthodes les plus utilisées. Une autre méthode intéressante est un croisement uniforme, cela veut dire qu'on choisit aléatoirement chaque gène d'un des deux parents. [5]

2.1.7 Méthode de Monte Carlo pour l'apprentissage par renforcement

La méthode de Monte Carlo a été implémentée pour l'apprentissage machine par renforcement. Ce dernier simule des épisodes aléatoires de séquences d'actions dans un arbre d'actions. Lorsque l'épisode atteint un nœud terminal, une récompense est retournée pour cette séquence d'action, ce qui fait varier le poids des nœuds, mettant en évidence que certains nœuds d'action sont plus favorables que d'autres.

Cet algorithme prend en entrée les paramètres de l'environnement dans lequel il sera appliqué, ainsi que des conditions d'arrêt de l'algorithme, soit le nombre d'itérations maximal ou un seuil d'acceptation basé sur les récompenses reçues.

La méthode de Monte Carlo se divise en 4 étapes répétées de manières itératives jusqu'à ce que l'on atteigne une condition d'arrêt spécifiée par l'utilisateur .

La première étape est celle de la sélection du prochain nœud à visiter, qui sera alors le départ exploratoire. Cette sélection se fait en parcourant l'arbre d'actions en choisissant celle avec la valeur obtenue la plus élevée jusqu'à ce qu'on arrive à un nœud feuille de l'arbre. Ce nœud feuille est alors choisi comme départ exploratoire, duquel l'algorithme va simuler un épisode aléatoire.

La deuxième étape de l'algorithme est celle de l'expansion du nœud sélectionné. Cette expansion ne fait que trouver les prochaines actions qui peuvent être réalisées à partir du noeud de départ et créer ses noeuds enfants à partir de ces actions.

La troisième étape de l'algorithme est celle de la simulation. À partir du nœud sélectionné, il y a une simulation aléatoire de l'épisode jusqu'à un nœud terminal. Rendu à ce nœud terminal, le résultat de cet épisode est calculé.

La dernière étape est alors de propager le résultat obtenu de la simulation jusqu'au nœud racine de l'arbre et en incrémentant le nombre de visites de chaque nœud parcouru lors de la simulation.

La condition d'arrêt de cette boucle à 4 étapes peut être une contrainte du nombre d'itérations et/ou un seuil acceptable pour le résultat obtenu.

À la mi-session, l'algorithme ainsi que son test de base étaient intégrés et fonctionnels. Depuis la mi-session, un prototype de parcours de graphe a été développé pour démontrer la méthode de Monte Carlo pour l'apprentissage par renforcement.

Des limitations de mémoire étaient présentes lors de l'implémentation du prototype à exécuter sur le microcontrôleur. Des optimisations ont donc été ajoutées, comme la diminution de la taille des structures de données utilisées, les nœuds d'action, ainsi que l'optimisation des tailles des variables en spécifiant des types plus précis et spécifiques à la méthode de Monte Carlo.

2.1.8 Méthodes statistiques

Plusieurs méthodes statistiques ont été implémentées dans la librairie afin d'évaluer les caractéristiques d'une distribution de points. Notamment, les 4 moments d'une distribution, soit la moyenne, la variance, le coefficient d'asymétrie et le kurtosis, ont été ajoutés.

De plus, la régression linéaire simple, une technique d'analyse statistique qui permet de modéliser la relation entre une variable dépendante et une variable indépendante en utilisant une droite de régression, a aussi été ajoutée à la librairie.

Finalement, la méthode de regroupement des K-Moyennes a été ajoutée afin de permettre aux utilisateurs de la librairie de regrouper des données en fonction de leurs

caractéristiques communes. Cet algorithme regroupe les points similaires en essayant de former des groupes les plus distincts possibles.

2.1.9 Décomposition LU

La décomposition LU est un algorithme utilisé pour séparer une matrice carrée en un produit de deux matrices plus simples à utiliser. L'une étant triangulaire supérieure et l'autre triangulaire inférieure. Cet algorithme est largement utilisé pour simplifier la résolution de systèmes d'équations à plusieurs variables. Cette méthode a été ajoutée à la librairie en se basant sur les opérations matricielles déjà existantes permettant de simplement appliquer cet algorithme sur une matrice carré.

Une limitation existe à cet algorithme ou la matrice d'entrée doit être inversible pour que l'algorithme puisse fonctionner. Cependant en pratique cette restriction est presque toujours respectée et donc l'algorithme est souvent utilisé pour, par exemple, effectuer de l'animation 3D ou encore choisir les recommandations à afficher pour un utilisateur en se basant sur le retour de celui-ci.

2.1.10 Méthode des différences finies

La méthode des différences finies est une technique d'approximation permettant d'évaluer la dérivée d'une fonction. Celle-ci permet d'approximer une première, seconde, troisième, etc. dérivée avec une précision de premier, second, troisième, etc. ordre. Le temps de calcul de cette méthode augmente grandement plus l'ordre de précision demandée s'accroît.

Dans la librairie, nous avons implémenté le calcul d'une approximation du gradient en utilisant la méthode des différences finies. Les équations décrivent les approximations de premier et second degrés.

$$Premier\ degré: f'(x) = \frac{f(x+h) - f(x)}{h}$$

$$Second\ degré: f'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

Dans les deux approximations, il est nécessaire de choisir une valeur de h petite afin d'avoir la meilleure approximation possible. La valeur de h est limitée par la précision à point flottant de la machine sur laquelle l'algorithme est exécuté. L'utilisation de l'une ou l'autre de ces approximations dépend de la forme de la fonction que l'on veut optimiser et le choix sera donc à l'utilisateur de spécifier laquelle prendre.

2.1.11 Sinus rapide et cosinus rapide

L'implémentation d'un sinus rapide et d'un cosinus rapide n'était initialement pas prévue. Cependant, suite à la demande du client, il a fallu implémenter ces méthodes afin d'accélérer chacune de ces fonctions mathématiques. Plusieurs méthodes ont été implémentées afin de satisfaire les besoins du client. Afin de centraliser chacune des options, elles ont été combinées en une seule, où l'utilisateur peut décider du degré de précision et de la performance.

Les implémentations réalisées peuvent être divisées en trois. Tout d'abord, pour des degrés de précision plus élevés, la technique d'approximation de Chebychev, combinée avec la méthode de multiplication de Horn est utilisée [2]. Ceci permet d'obtenir une très haute précision pour un sinus ou un cosinus, mais au coût de faire des multiplications en point flottant, ce qui est plus lent.

Afin de calculer des degrés de précision plus bas, une table trigonométrique est utilisée. Ceci fait en sorte que des calculs en point flottant ne sont plus nécessaires, permettant alors d'accélérer les calculs. Cependant, puisque la précision peut être grandement influencée par les valeurs dans la table, une interpolation est faite pour augmenter la précision.

Finalement, une méthode de calcul du sinus et cosinus d'un angle en point fixe a aussi été ajoutée. Pour certains algorithmes, il est possible de déterminer en point fixe l'angle qui devra être calculé, évitant alors l'usage de virgules flottantes. Cette méthode permet alors d'obtenir une accélération maximale lorsqu'elle est combinée avec une table trigonométrique. La précision est conservée en effectuant une interpolation sur les valeurs de la table.

2.1.12 Pipelines

Des pipelines de validations ont été développés pour cadrer le développement des algorithmes. Ces pipelines ont été implémentés à l'aide des outils fournis par Github. Le premier pipeline vérifie la compilation de tous les algorithmes, vérifie le formatage des fichiers de code et vérifie que tous les tests des algorithmes donnent les résultats attendus.

Ce pipeline est exécuté automatiquement lorsqu'une requête de fusion est mise à jour ou lorsque le code dans la branche principale est modifié. Cela permet de déterminer si des requêtes de fusions brisent des fonctionnalités de la libraire et s'assure de maintenir une librairie avec des normes de formatage. Ce pipeline est inclus dans la branche principale.

Le deuxième pipeline a été développé, mais il n'est pas inclus dans la branche principale en raison d'un conflit de permission. Ce pipeline s'occupe de compiler la documentation de la librairie qui se trouve dans un fichier Latex et de l'ajouter au dépôt Github. L'étape de la compilation s'exécute lorsqu'une requête de fusion contenant une modification à la documentation est mise à jour. Une fois que la requête de fusion est acceptée, le pipeline ajoute la documentation compilée au dépôt Github.

2.1.13 Documentation

Différentes méthodes de documentation ont été utilisées tout au long du projet, afin de rendre celle-ci plus facile à utiliser.

Une première forme de documentation utilisateur qui a été écrite est celle d'un document centralisé, en format PDF, dans lequel chaque algorithme développé dans la

librairie possède une description mathématique plus détaillée. Ce document parcourt donc le fonctionnement de chaque algorithme, donnant une vue de plus haut niveau de ceux-ci.

Ensuite, pour la documentation système, dans les fichiers sources des algorithmes implémentés, chaque fonction est décrite par quelques lignes de documentation, énumérant le but de la fonction ainsi que les différents paramètres d'entrée et de sortie de celle-ci. De cette manière, le code est plus compréhensible pour un utilisateur qui souhaite comprendre plus en détail l'implémentation de chaque algorithme.

De plus, chaque prototype d'algorithme est accompagné d'un fichier d'instructions, une documentation utilisateur, dans lequel sont explicitées des indications claires sur la procédure à suivre afin d'exécuter le prototype. Ce fichier dénote également le fonctionnement du prototype. De cette manière, en suivant les instructions de ce fichier, l'utilisateur peut, par soi-même et aisément, tester un prototype d'un algorithme sur un microcontrôleur.

2.2 Quelques diagrammes de classes ou de modules

Il est crucial de souligner que le client exige que la librairie ne soit pas liée à d'autres librairies externes, à l'exception de la librairie standard de C pour la création d'algorithmes. Ainsi, la plupart des algorithmes sont conçus pour être autonomes et ne dépendent pas des autres modules de la librairie. La structure des modules est décrite dans la figure 1 où les flèches représentent les dépendances entre ces derniers.

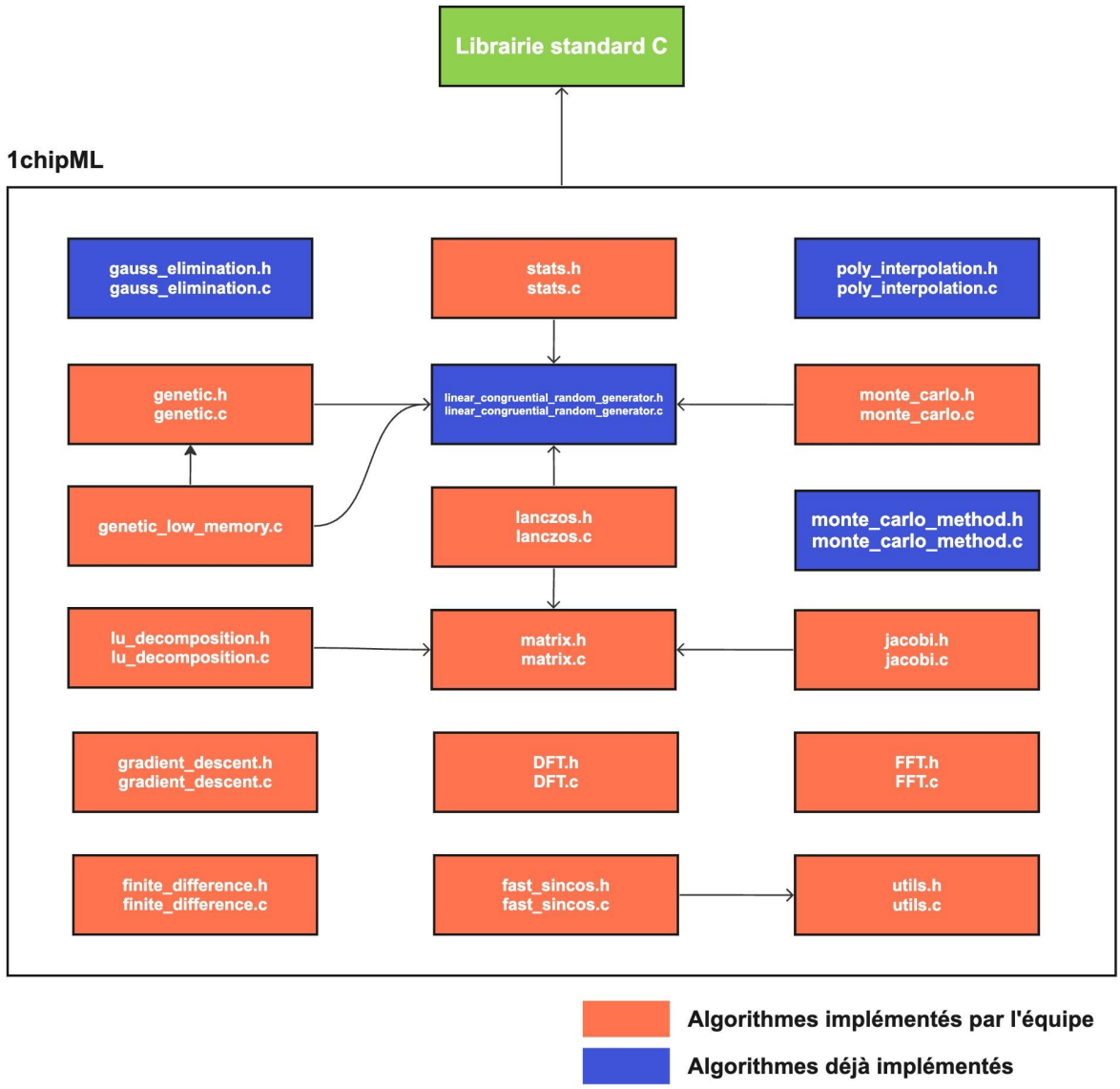

Figure 1 : Diagramme des modules de la librairie 1chipML

En outre, les seuls modules de la librairie qui ne sont pas considérés comme des algorithmes à part entière sont les modules "util" et “matrix”. Ces derniers regroupent plusieurs constantes ainsi que plusieurs méthodes qui sont fréquemment utilisées par les algorithmes, tels que la multiplication de matrices, la transposition de matrices et la mise à l'échelle de vecteurs.

Du fait que la librairie a été développée dans le langage C, le concept de classe n'existe pas et aucune structure spécifique n'a été mise en place pour implémenter les algorithmes. Les types de données fournis par le langage ont été les seuls utilisés durant le développement, ce qui ne permet pas de fournir un diagramme de classes.

2.3 Diagramme d'états et/ou d'interface usager

Puisque la tâche à compléter pour notre client est la création d'une librairie implémentant divers algorithmes mathématiques sur un microcontrôleur. La notion d'état et d'interface usager ne s'applique pas à notre projet. En effet, chacun de nos algorithmes est implémenté par une fonction qui s'exécute de manière séquentielle et retourne une réponse. Il n'y a donc pas de notion d'état dans ceux-ci. Ensuite, puisque notre projet consiste en la création d'une librairie, nous fournissons des interfaces logicielles, c'est-à-dire des fonctions qui peuvent être appelées par les utilisateurs de la librairie. La notion d'interface usager ne s'applique donc pas non plus.

2.4 Interfaces logicielles et/ou matérielles importantes

Du côté logiciel, les interfaces importantes pour le développement de notre projet sont les diverses définitions des algorithmes que nous avons implémentées.

Premièrement, chaque algorithme doit utiliser un type défini spécifiquement pour celui-ci. Par exemple, au lieu d'utiliser le type « double », chaque algorithme doit définir son propre type équivalent comme « lanczos_real » pour l'algorithme de Lanczos. Cela est fait afin d'éviter des problèmes reliés à la différence de taille et de précision de certains types tels que « float » et « double » sur différent modèle de microcontrôleur.

Ensuite, chaque algorithme développé expose aux utilisateurs de la librairie une fonction afin d'utiliser l'algorithme. Les paramètres de ces fonctions sont variables et dépendent du type d'algorithme qui est implémenté.

Pour les algorithmes de FFT et de DFT, ceux-ci permettent de transformer des données du domaine temporel au domaine fréquentiel. Les fonctions prennent en paramètre deux tableaux dont la taille est une puissance de deux, le premier étant la composante réelle des données et le second, leur composante complexe. Le résultat de l'algorithme sera renvoyé à l'utilisateur à travers ces deux mêmes tableaux.

Pour l'algorithme de Lanczos, celui-ci permet d'appliquer l'algorithme sur une matrice symétrique passé en paramètre d'entrée. Ensuite, l'algorithme décompose cette matrice en deux matrices T et V qui sont retournées à l'utilisateur.

Pour l'algorithme de Jacobi, celui-ci permet de déterminer les valeurs et vecteurs propres d'une matrice fournie en entrée. La matrice d'entrée doit être carrée et symétrique. Les valeurs propres et vecteurs propres seront retournés à l'utilisateur dans deux matrices distinctes.

Pour l'algorithme Monte Carlo, celui-ci permet d'itérer à travers un arbre d'actions pour déterminer la meilleure action à poser dans un contexte particulier. L'utilisateur doit donc fournir ce contexte et les actions qui peuvent en découler. Plus précisément, l'utilisateur doit décrire le jeu auquel il souhaite appliquer l'algorithme de Monte Carlo. Ces précisions sont les spécifications quant au plateau de jeu, les différentes actions

possibles du jeu, l'exécution de l'action, le retrait de l'action et le résultat obtenu du jeu. De plus, l'utilisateur doit spécifier les conditions d'arrêts de l'algorithme, soit le seuil acceptable du résultat obtenu de l'algorithme, ou bien le nombre d'itérations maximal et minimal.

Pour l'algorithme génétique, celui-ci permet d'optimiser une fonction fournie en paramètres et retourne les valeurs optimales de celle-ci. Il est possible de spécifier plus exactement le comportement de l'algorithme à l'aide d'autres paramètres comme la taille des générations, la chance de mutations, etc.

Pour l'algorithme de la descente du gradient, celui-ci permet d'optimiser une fonction fournie en paramètre et retourne le minimum de celle-ci. Certains paramètres supplémentaires sont fournis à la fonction afin de spécifier la précision de celle-ci et pour limiter le nombre d'opérations.

Pour l'algorithme des différences finies, celui-ci permet de calculer une approximation du gradient d'une certaine fonction spécifiée par l'utilisateur. Certains paramètres supplémentaires sont fournis à la fonction afin de spécifier la précision de celle-ci et pour choisir le type d'approximation.

Pour les divers algorithmes statistiques, ceux-ci permettent d'obtenir plusieurs informations pertinentes telles que la moyenne, variance, etc. sur un jeu de données. Il est possible d'utiliser individuellement chaque méthode ou d'appeler une méthode globale qui calcule toutes ces valeurs d'un coup.

Pour l'algorithme de décomposition LU [6], celui-ci permet de factoriser une matrice en une matrice triangulaire supérieure et une matrice triangulaire inférieure.

Pour les implémentations du sinus rapide et du cosinus rapide, ceux-ci permettent d'obtenir une approximation de chaque fonction mathématique. L'utilisateur peut appeler chacune des fonctions individuellement, avec un paramètre supplémentaire pour spécifier le degré de précision désiré.

Pour ce qui est des interfaces matérielles, celles-ci incluent des microcontrôleurs Arduino qui seront utilisés afin de créer des prototypes démontrant certaines applications réelles des algorithmes sur un microcontrôleur. Il sera donc nécessaire d'utiliser une communication sérielle entre l'Arduino et un ordinateur afin de récupérer les résultats produits par le Arduino lors de l'exécution des algorithmes.

Des interfaces logicielles ont été développées en C++ et en Python afin de standardiser les communications entre le Arduino et l'ordinateur. Dans celles-ci, on fournit des appels bloquants permettant de recevoir et de transmettre n'importe quel type de données sous la forme d'octets sur un port sériel.

2.5 Description d'outils, librairies ou cadre de travail (*framework*) utilisés

Dans le cadre du projet plusieurs outils ont été utilisés pour faciliter la tâche de développement. Le premier outil utilisé pour la création des prototypes est l'éditeur de code Arduino IDE. Ce logiciel permet de facilement exécuter le code sur les microcontrôleurs Arduino Uno qui étaient fournis. Un deuxième outil qui a facilité la tâche de développement est le logiciel ClangFormat.[7] Ce logiciel permet de formater les fichiers de code en fonction des standards établis. Cela permet de maintenir un niveau d'uniformité au dépôt, ce qui favorise l'utilisation de la librairie. Le dernier outil utilisé pour faciliter le développement est la suite de pipelines automatiques disponible sur Github. Ces pipelines nous ont permis de nous assurer que les requêtes de fusion ne brisent pas les solutions déjà implémentées.

Afin d'exécuter notre code, nous avions besoin de compilateurs. Le compilateur avr-gcc est celui qui est utilisé pour compiler le code en exécutable qui peut être exécuté sur des microcontrôleurs. Ce compilateur vient préinstallé à l'intérieur de Arduino IDE.

Plusieurs outils ont également été utilisés pour aider la validation des implémentations. Le premier outil est Valgrind, cet outil est très efficace pour détecter les fuites de mémoire présentes dans les logiciels qui ont été développés avec le langage de programmation C. Le deuxième outil utilisé est l'option AddressSanitizer présente sur le compilateur GCC [8]. Cette option permet de détecter lorsqu'il y a une opération invalide sur une case mémoire. Le dernier outil utilisé est une fonction développée par Adafruit qui permet de détecter la taille de la quantité de mémoire de la pile qui est disponible lorsqu' on l'exécute sur un microcontrôleur. Cela a permis de vérifier que les algorithmes ne consommaient pas trop de mémoire.

2.6 Quelques méthodes de tests appliquées pour valider la solution

2.6.1 Validation des algorithmes

Afin de valider les différents algorithmes implémentés, il y a d'abord un test de base qui est effectué sur un ordinateur pour valider l'algorithme. Puis, un prototype, correspondant à une application réelle de l'algorithme, est ensuite implémenté sur un Arduino pour vérifier que l'algorithme fonctionne correctement sur les microcontrôleurs.

2.6.2 Transformée de Fourier rapide (FFT) et transformée de Fourier discrète (DFT)

Afin de valider le fonctionnement de la DFT et de la FFT, une matrice et son résultat ont été calculés à l'avance et sont fournis en entrée aux deux algorithmes. Il est possible de déterminer si les algorithmes fonctionnent si leur résultat correspond au résultat attendu. De plus, la DFT et la FFT sont utilisées pour vérifier le fonctionnement de l'un et de l'autre, à partir de valeurs aléatoires. En d'autres mots, des données générées aléatoirement sont envoyées vers les deux algorithmes et leurs résultats sont ensuite comparés. Étant donné qu'ils effectuent les mêmes types de calculs, leurs résultats

devraient être identiques. Également, le prototype de convolution d'image avec FFT a été implémenté pour valider le fonctionnement de l'algorithme sur microcontrôleur. En effet, des données seront envoyées vers un Arduino par communication sérielle afin qu'il puisse appliquer une FFT sur celles-ci. Ensuite, les résultats sont renvoyés à l'utilisateur et seront utilisés pour analyser le résultat de la convolution.

2.6.3 Lanczos

Le premier test consiste en une vérification formelle de l'algorithme. Pour ce faire, une matrice a été sélectionnée et les résultats attendus de l'algorithme ont été calculés manuellement. Le test consiste donc à comparer les résultats obtenus suite à l'exécution de l'algorithme à ceux qui ont été précédemment calculés.

Le prototype consiste à appliquer une analyse en composante principale (PCA) sur un jeu de données utilisé pour entraîner un modèle à effectuer de la classification. Suite à l'application de la PCA, ces données peuvent par la suite être représentées dans un espace en 2 dimensions tout en gardant la majorité des informations de l'espace initial.

2.6.4 Jacobi

Tout d'abord, le premier test consiste à vérifier qu'étant donné une matrice quelconque dont les valeurs et vecteurs propres sont connus, l'algorithme retourne le bon résultat. Il est important de mentionner que ce test est validé dans le cas de la version cyclique de l'algorithme et dans le cas de la méthode traditionnelle.

Le prototype consiste à effectuer une réduction de dimensionnalité pour compresser une image à l'aide d'une PCA. En effet, en prenant la matrice de covariance d'une image et en calculant les valeurs et vecteurs propres de celle-ci, il est possible d'enlever une grande partie des données sans compromettre l'information qui se trouve dans l'image. La matrice de covariance est envoyée à un microcontrôleur qui exécute l'algorithme de Jacobi pour ensuite renvoyer la réponse à un ordinateur qui affiche l'image. L'image est ensuite validée qu'elle a bien été compressée.

2.6.5 Descente du gradient

Pour ce qui est de la méthode de la descente du gradient conjuguée. Afin de valider son bon fonctionnement, le premier test consiste à demander à l'algorithme de déterminer le minimum d'une fonction simple telle que $f(x, y) = (x - 4)^2 + (y + 3)^2$ et de voir si l'algorithme nous retourne le bon minimum de (4, -3) selon la tolérance spécifiée.

Le prototype choisi pour cet algorithme est la descente du gradient afin de déterminer le polynôme de degré N qui décrit un ensemble de points pour par la suite extrapoler ces points dans le futur. L'information calculée par le Arduino est ensuite envoyée à un ordinateur ou un programme Python qui s'occupe de valider celle-ci en effectuant lui aussi une régression polynomiale et en comparant les résultats.

2.6.6 Algorithmes génétiques

Le premier test de l'algorithme génétique est très similaire à celui de la descente du gradient. Cet algorithme est utilisé pour trouver l'optimum global d'une équation mathématique. Ainsi, pour valider le fonctionnement de celui-ci, la fonction suivante est fournie en entrée: $f(x, y) = abs(4x - e^{(2*y)} + 2)$.

Les valeurs de x et y sont bornées entre 0 et 1 pour simplifier le test. On vérifie ensuite que l'algorithme retourne une solution qui est valide selon la tolérance. Puisque cet algorithme n'a pas la garantie de trouver la solution exacte. La tolérance est requise pour vérifier que la solution retournée est suffisamment proche

En ce qui concerne le prototype implémenté sur le microcontrôleur, l'algorithme doit être en mesure d'utiliser une fonction polynomiale générique pour représenter un nuage de points et doit trouver les paramètres qui font correspondre la « spline » à la série de points. La méthode de validation sera la même que celle de la descente du gradient, nous utiliserons un programme Python pour valider nos résultats à l'aide d'une régression polynomiale.

2.6.7 Méthode de Monte Carlo pour l'apprentissage par renforcement

Pour valider le fonctionnement de l'algorithme Monte Carlo, deux tests sont implémentés. Le premier test est celui du jeu « Tic-Tac-Toe ». Il consiste en l'implémentation du jeu et des différentes fonctions qui sont spécifiques à celui-ci, comme vu dans la section 2.4. Ces fonctions sont passées en paramètres à celle exécutant l'algorithme Monte Carlo. Ensuite, le test vérifie si à partir d'un état connu du jeu « Tic-Tac-Toe » et dont le prochain état favorable est connu également, le résultat obtenu de l'exécution de l'algorithme est celui attendu. Le second test, le prototype, pour vérifier le bon fonctionnement de l'algorithme Monte Carlo est celui de la recherche du plus court chemin pour parcourir un graphe. Il consistera donc à passer à la fonction la description d'un graphe ainsi que les fonctions décrivant les différentes actions du parcours du graphe. Ensuite, il vérifiera si le parcours retourné est celui le plus court en comparant avec la valeur juste, connue d'avance.

2.6.8 Méthodes statistiques

Pour valider le fonctionnement des différentes méthodes statistiques, plusieurs tests ont été développés. Tout d'abord, afin de tester individuellement chacune des méthodes (moyenne, variance, corrélation), une distribution de données, dont les différents moments et valeurs statistiques sont connus, a été créée. Ensuite, le résultat retourné par les méthodes est comparé aux valeurs connues pour valider le fonctionnement des fonctions. Ensuite, un prototype a été établi pour valider l'algorithme de K-Moyennes. Un programme python s'occupe de générer des points qui sont regroupés en K groupes. Ensuite, ces points sont envoyés à un Arduino qui exécute l'algorithme de K-Moyennes sur les points envoyés. Ensuite, lorsque les assignations sont renvoyées, elles sont validées en affichant un graphique des assignations aux différents groupes.

De plus, les positions des centroïdes sont comparées aux valeurs obtenues par la librairie *Scikit-learn* pour s'assurer qu'elles sont valides.

2.6.9 Décomposition LU

Pour valider l'algorithme de décomposition LU, deux tests ont été développés. Le premier test est un test simple qui compare les résultats obtenus à l'application de l'algorithme par rapport à des résultats calculés manuellement. Ce test permet donc de vérifier que l'algorithme peut être appliqué sur une petite matrice et qu'il retourne des résultats cohérents. Le deuxième test consiste en l'application d'un système de recommandation. À partir d'une matrice de retour d'information des utilisateurs partiellement remplie, l'algorithme de décomposition LU et l'algorithme de différences finies sont utilisés pour compléter les informations manquantes. Pour vérifier que ces algorithmes sont fonctionnels, il est donc possible de vérifier que la matrice finale est cohérente avec la matrice initiale et que les informations manquantes ont été ajoutées.

2.6.10 Méthode des différences finies

Pour ce qui est de la méthode des différences finies, afin de valider son bon fonctionnement, le premier test consiste à demander à l'algorithme de déterminer le gradient d'une fonction simple telle que $f(x, y) = (x - 3.5)^2 + (y + 4)^2$ au point (3, 5) de voir si l'algorithme nous retourne le bon gradient de (-1, 18) selon la tolérance spécifiée.

Pour ce qui est du prototype sur Arduino, l'algorithme des différences finies a été validé par son utilisation dans le prototype du système de recommandation de l'algorithme LU.

2.6.11 Sinus rapide et cosinus rapide

Afin de valider l'algorithme d'un sinus rapide et d'un cosinus rapide, un premier test a été établi afin de mesurer la précision. En effet, afin de vérifier le fonctionnement de l'implémentation, il faut s'assurer que le résultat obtenu ne diffère pas trop de l'implémentation originale. Il a alors fallu calculer les erreurs relatives et les erreurs absolues afin d'identifier si les algorithmes étaient acceptables.

Ensuite, un prototype a été créé afin de mesurer l'accélération obtenue par les implémentations, selon le degré de précision. Ce prototype est essentiel aux implémentations de sinus et cosinus, car il permet d'illustrer les avantages de ceux-ci par rapport à leur implémentation traditionnelle.

2.6.12 Développement opérationnel

Un pipeline de validation se déclenchant à chaque fois qu'une requête de fusion est créée a aussi été mis en place. Le pipeline de validation est constitué de plusieurs étapes, chacune effectuant une tâche spécifique. La première étape consiste à compiler le code et à s'assurer qu'il ne contient aucune erreur ou avertissement. La deuxième étape est une vérification de la qualité du code en termes de respect des normes de

codage. Pour ce faire, l'outil d'analyse statique de code ClangFormat [9] qui vérifie que le code est conforme aux conventions et standards de codage établis dans notre équipe.

Enfin, la troisième étape du pipeline de validation consiste à exécuter l'ensemble des tests automatisés, pour s'assurer que toutes les fonctionnalités existantes fonctionnent toujours correctement. Si une erreur est détectée à l'une des étapes, la requête de fusion ne peut être fusionnée au code principal, obligeant les développeurs à corriger le problème avant de poursuivre. En mettant en place un pipeline de validation automatisé, il est possible de garantir que chaque modification apportée au code est bien vérifiée et validée avant d'être intégrée dans le code source principal, réduisant ainsi les risques d'erreur et garantissant une qualité de code élevée.

3. Gestion du projet

3.1 Identification des tâches principales du projet

Le projet consiste en l'implémentation d'algorithmes communs en intelligence artificielle à des fins d'utilisation sur des microcontrôleurs dans une librairie à code ouvert.

Le projet se divise donc en plusieurs tâches indépendantes entre elles, soit l'implémentation des différents algorithmes, et ensuite en sous-tâches dépendantes spécifiques à chaque algorithme, comme les tests, la documentation, les prototypes et l'optimisation.

3.1.1 Implémentation

Pour ce qui est de l'implémentation des algorithmes, voici une liste détaillée des tâches principales:

- Transformée de Fourier rapide
- Jacobi
- Lanczos
- Génétique
- Descente du gradient
- Monte Carlo pour l'apprentissage par renforcement
- Des méthodes statistiques
- Décomposition LU
- Méthodes des différences finies
- Sinus rapide et cosinus rapide

Pour chacune des tâches définies dans la liste ci-dessus, il faut implémenter l'algorithme pour que ce dernier puisse s'exécuter sur un microcontrôleur.

3.1.2 Recherche

L'implémentation des algorithmes nécessite avant tout une bonne compréhension de ceux-ci. Puisque ces algorithmes n'étaient pas maîtrisés préalablement au projet, une tâche importante à réaliser pour ce dernier était celle de recherche sur les algorithmes à implémenter. Après s'être renseigné davantage sur la théorie derrière ceux-ci, une seconde tâche importante de recherche est celle sur les différentes implémentations possibles des algorithmes et leurs variantes. Ces tâches de recherche permettront d'être suffisamment outillés pour entamer les tâches d'implémentation des algorithmes. Voici donc une liste des tâches de recherche qui constituent le projet:

- Recherche sur les notions théoriques des différents algorithmes
- Recherche sur les différentes implémentations des algorithmes et leurs variantes

3.1.3 Optimisation

Après l'implémentation des algorithmes listés plus haut, puisque ces derniers sont destinés à s'exécuter sur des microcontrôleurs, des contraintes de mémoire doivent être prises en considération. Cela dit, une tâche importante à réaliser dans ce projet est l'optimisation de la mémoire utilisée par chaque algorithme. De plus, afin de rendre les algorithmes d'intelligence artificielle plus performants, un facteur important est celui du temps, il y a donc l'optimisation de la complexité de temps qui s'ajoute également aux tâches importantes du projet. Voici donc une liste des tâches d'optimisation:

- Optimisation de la complexité d'espace pour chaque algorithme
- Optimisation de la complexité de temps pour chaque algorithme

3.1.4 Prototype

Une tâche importante du projet est celle de réaliser des prototypes pour chaque algorithme. Effectivement, les prototypes permettent aux utilisateurs de la librairie de mieux comprendre les applications de celle-ci, ainsi que de montrer l'utilité de celle-ci. Les prototypes sont donc des exemples d'application concrets des algorithmes. Ces prototypes doivent pouvoir être exécutés sur un microcontrôleur.

Voici donc une liste détaillée des tâches de prototypage ainsi que les différents algorithmes associés:

- Apprendre le jeu «Tic-Tac-Toe» (Monte Carlo pour l'apprentissage par renforcement)
- Parcours de graphe pour trouver le plus court chemin (Monte Carlo pour l'apprentissage par renforcement)
- Convolution d'image (Transformée de Fourier rapide)
- L'analyse d'images en composantes principales (Méthode de Jacobi et de Lanczos)
- Courbe d'ajustement (Algorithme génétique et descente du gradient)
- Système de recommandations (Décomposition LU et différence finie)

- Regroupement de données (Méthodes statistiques)

3.1.5 Documentation

Une tâche importante du projet est celle de la documentation de chaque algorithme implémenté. Effectivement, puisque c'est un projet à code ouvert et qu'il aura une continuité à l'extérieur de notre cours, il est très important de documenter ce dernier. Non seulement est-il nécessaire de documenter les implémentations des algorithmes, les prototypes développés doivent l'être aussi. En effet, ces prototypes servent à la compréhension de la communauté du projet à code ouvert et rendent cette librairie plus simple d'usage. La documentation consiste en la génération d'un fichier .pdf avec la description des algorithmes, de leur implémentation et des méthodes d'utilisation de ceux-ci. La forme de la documentation est décrite davantage dans la section 2.1.13 du présent rapport.

S'ajoutent donc aux tâches, celles de la documentation:

- Documentation des implémentations des algorithmes
- Documentation des prototypes pour chaque algorithme

3.2 Répartition des tâches et responsabilités dans l'équipe

L'équipe a commencé le projet en attribuant des rôles à chaque membre pour faciliter le déroulement du projet.

Florence est responsable des communications ainsi que de prendre des notes lors des réunions, c'est-à-dire qu'elle s'occupe de toute communication qui se déroule entre notre équipe.

Nicholas a le rôle de gestionnaire de l'environnement de travail. Il a créé et gère un canal Slack pour simplifier la communication avec le client lorsque des membres de l'équipe ont des interrogations sur la méthode d'accomplissement des tâches. Le deuxième élément de ce rôle était d'implémenter un standard de programmation décidé par l'équipe.

Paul-André a le rôle de gestionnaire des tests. L'équipe d'Ericsson qui contribue à ce projet a mis en place quelques tests qui s'exécutent par le biais d'un Makefile. Il est donc responsable de mettre une procédure uniforme pour le développement des tests. Cela augmente grandement la facilité de compréhension des tests par les tiers puisque la structure de chaque fichier de test est la même. Un utilisateur de la libraire peut alors facilement modifier les tests en fonction de leurs besoins.

Félix-Antoine est le responsable des requêtes de fusion. Tous les membres de l'équipe participent à la revue de code, mais il a la responsabilité finale de l'approbation de la requête de fusion ainsi que de s'assurer que le client est au courant de l'ouverture de la

requête. Il est aussi responsable de faire un suivi des requêtes pour s'assurer que celles-ci ne bloquent pas.

Alexis est l'animateur, c'est-à-dire qu'il recueille les idées de tout le monde afin de créer les plans de rencontre pour nos réunions hebdomadaires en équipe. Il s'occupe de gérer nos réunions afin que tous les points soient abordés. Lors de nos rencontres avec le client, il s'occupe de partager les idées de l'équipe et de poser nos questions. Son rôle est très utile pour faciliter le déroulement des rencontres.

Gaya est le responsable du temps et du déroulement des tâches. Il s'assure que le temps pris par chaque point de notre plan de rencontre ne dépasse pas le temps qui lui est alloué. On minimise alors les chances de débordement de rencontre. Il est également le responsable de la gestion de notre tableau Kanban. Il s'assure que l'échéancier est respecté et modifie le tableau s'il y a des imprévus lors de l'accomplissement des tâches ou si le client a une demande supplémentaire.
Le tableau 1 montre la division des tâches qui ont été énoncées à la section 3.1 entre les membres de l'équipe.

Tableau 1: Tâches à accomplir pour chaque membre de l'équipe

Membres de l'équipe	**Tâches à accomplir**
Florence	● Implémentation de l'algorithme Monte Carlo pour l'apprentissage par renforcement (100 heures) ● Apprendre le jeu Tic-Tac-Toe (Monte Carlo pour l'apprentissage par renforcement) (45 heures) ● Parcours de graphe pour trouver le plus court chemin (Monte Carlo pour l'apprentissage par renforcement) (40 heures)
Nicholas	● Implémentation de l'algorithme génétique (85 heures) ● Courbe d'ajustement (Algorithme génétique) (50 heures) ● Implémentation des pipelines (30 heures)
Paul-André	● Implémentation d'un algorithme de transformée de Fourier rapide (85 heures) ● Convolution d'image (Transformée de Fourier rapide) (45 heures) ● Sinus rapide et cosinus rapide (50 heures)
Félix-Antoine	● Implémentation d'un algorithme de Lanczos (95 heures) ● L'analyse d'images en composantes principales (Lanczos) (45 heures) ● Implémentation de l'algorithme de Décomposition LU (40 heures)

Alexis	● Implémentation de la technique d'optimisation de descente du gradient (85 heures) ● Courbe d'ajustement (Descente du gradient) (40 heures) ● Implémentation de l'algorithme des différences finies (30 heures)
Gaya	● Implémentation de l'algorithme de Jacobi (95 heures) ● L'analyse d'images en composantes principales (Jacobi) (45 heures) ● Implémentation de diverses méthodes statistique (45 heures)
Tous les membres	● Recherche sur les notions théoriques des différents algorithmes. (5 heures par membre) ● Recherche sur les différentes implémentations des algorithmes et leurs variantes. (10 heures par membre) ● Optimisation de la complexité d'espace pour chaque algorithme implémenté (10 heures par membre) ● Optimisation de la complexité de temps pour chaque algorithme implémenté (10 heures par membre) ● Recherche sur l'implémentation de l'algorithme du bandit manchot (5 heures) ● Documentation des implémentations des algorithmes (5 heures par membre) ● Documentation des prototypes pour chaque algorithme (10 heures par membre) ● Rencontres avec le client (10 heures par membre)

3.3 Principales difficultés de gestion rencontrées durant la deuxième partie du projet

Il y a plusieurs difficultés de gestion que l'équipe a rencontrées durant la deuxième moitié du projet. La première difficulté fut présente tout au long du projet. Il y a eu des requêtes de fusion qui ont été ouvertes tout au long de la session et il fallait alors s'assurer que tous les membres participent à la revue de code. De plus, l'équipe devait obtenir l'approbation du client avant de pouvoir fermer la requête. Il y a alors des délais possibles à gérer pour donner le temps suffisant aux membres et aux clients de participer à cette revue.

La deuxième difficulté était de trouver des blocs de temps ou tout le monde était disponible en dehors du temps déjà alloué pour le projet. Il a été extrêmement difficile pour l'équipe de trouver des blocs de temps où tous les membres étaient disponibles, il a donc fallu créer des plus petites rencontres avec des sous-groupes de l'équipe.

Certaines tâches telles que le développement des prototypes des algorithmes nécessitait des parties de code développées par d'autres membres de l'équipe. Le premier exemple de ça est le prototype de l'algorithme de Lanczos. Le prototype de cet algorithme était d'effectuer une analyse des composantes principales sur des images pour pouvoir ensuite les classifier. Ce prototype nécessite également l'algorithme de Jacobi. Il fallait donc prioriser le développement de l'algorithme de Jacobi pour qu'il ne devienne pas un bloquant du développement du prototype. Un deuxième exemple est le prototype de l'algorithme de décomposition LU. Cet algorithme est utilisé pour créer un système de recommandation et nécessite la méthode des différences finies. Encore une fois il fallait développer rapidement la méthode des différences finies pour pouvoir accorder suffisamment de temps pour le développement du prototype de l'algorithme de la décomposition LU. L'équipe a toutefois été en mesure de respecter tous les délais qui ont été définis en début de projet.

4. Conclusion

4.1 Retour sommaire sur le travail complété et ce qui n'a pu être réalisé (Q3.5)

De nombreux algorithmes ont pu être complétés. Parmi ceux-ci, on peut retrouver l'algorithme de la transformée de Fourier rapide, l'algorithme de Jacobi, l'algorithme de Lanczos, l'algorithme génétique, la descente du gradient, l'algorithme de Monte Carlo pour l'apprentissage par renforcement, la décomposition LU, la méthode des différences finies et diverses méthodes statistiques. Cependant, par souci de remettre un travail de qualité, l'algorithme du bandit manchot n'a pas été implémenté. Bien qu'il aurait été possible de concevoir cet algorithme, cela n'aurait pas laissé le temps à l'équipe de bien valider les autres fonctionnalités, ce qui aurait diminué la qualité du travail réalisé. Néanmoins, les objectifs initialement fixés ont été atteints. De plus, différentes méthodes supplémentaires aux algorithmes principaux ont été implémentées. Par exemple, une méthode rapide de calcul de sinus et de cosinus ainsi qu'une méthode de communication sérielle ont été ajoutées au projet.

De plus, divers tests et prototypes ont été ajoutés pour valider le fonctionnement de chaque algorithme sur microcontrôleur. Ces prototypes permettent en plus d'illustrer des cas d'utilisation des algorithmes, permettant aux utilisateurs de mieux connaître leur utilité. Également, une documentation complète de chaque algorithme a été réalisée pour assurer la compréhension de chaque implémentation. Finalement, des pipelines ont été ajoutés au projet pour assurer la qualité de son développement, même après la fin de notre implication avec ce dernier.

4.2 Causes des succès et difficultés rencontrées (Q3.6)

De nombreuses raisons ont permis à notre équipe de terminer le projet correctement et dans les délais imposés.

Premièrement, l'ensemble des tâches pouvaient être développées de manière indépendante. Puisque les algorithmes n'avaient pas de dépendances entre eux, nous avons pu développer ceux-ci en parallèle de manière à accélérer le travail. Cette indépendance nous a permis d'accélérer le processus de développement puisque les algorithmes simples pouvaient être fusionnés dans la branche principale rapidement alors que les algorithmes plus complexes pouvaient prendre plus de temps sans bloquer les tâches subséquentes.

Deuxièmement, le client était présent et nous a soutenus tout au long du projet. En effet, une plateforme de discussion a été mise en place avec le client dès le début du projet pour communiquer avec lui et lui poser nos questions lorsque nécessaire. Le client était donc facilement accessible par cette plateforme et répondait à nos questions rapidement. De plus, des rencontres bimensuelles étaient organisées en présentiel dans les bureaux de la compagnie. Lors de ces rencontres, le client nous apportait des commentaires pertinents et des références utiles pour le développement des algorithmes. Les critères du projet étaient clairs et bien définis. Ceci nous a donc permis de bien préparer celui-ci pour nous assurer que l'équipe soit en mesure de terminer les tâches dans les délais requis.

Troisièmement, notre équipe était bien structurée puisque nous ne travaillons pas ensemble pour la première fois. En effet, nous avons déjà fait plusieurs projets ensemble dans le passé ce qui nous permet donc de bien connaître les forces et les faiblesses de chaque membre de notre équipe. Ainsi nous avons été en mesure, dès le début du projet, de clairement définir les rôles de chaque membre en fonction des forces de chacun. De plus, plusieurs plateformes de communication étaient à notre disposition nous permettant de facilement communiquer avec les autres membres de l'équipe.

Malgré le succès du projet, l'équipe a tout de même rencontré certaines difficultés qui ont complexifié le projet. Premièrement, les algorithmes de la librairie devaient être développés pour pouvoir être exécutés sur des microcontrôleurs avec une mémoire limitée. Il était donc important, dès le départ, d'effectuer une implémentation qui prenait en considération ce critère limitant. Certains algorithmes ne proposaient pas d'implémentation faible en mémoire et il fallait donc les adapter de manière à ce qu'ils puissent s'exécuter sur des microcontrôleurs.

Deuxièmement, puisque le projet est à source ouverte, il est important de s'assurer que le code présenté est uniforme. En effet, une présentation uniforme à l'aide d'un standard de programmation simplifie la lecture du code par les nouveaux utilisateurs et permet donc aux nouveaux contributeurs de participer plus facilement. Pour ces

raisons, l'équipe devait donc s'assurer de mettre en place un standard de programmation et de le faire respecter.

Troisièmement, puisque nous avons dû travailler sur un projet relativement nouveau, nous avons dû mettre en place les bases de celui-ci. En effet, une première ébauche de l'architecture nous a été présentée représentant les requis du client, mais nous avons dû définir de nombreux éléments pour nous assurer du succès du projet. Par exemple, nous avons mis en place un standard de programmation, un pipeline de validation pour nous assurer que les nouvelles demandes de fusion respectent les critères. Nous avons aussi dû définir l'architecture des fichiers du projet ainsi que les fichiers standards des prototypes.

5. Apprentissage en continu (Q12)

5.1 Lacunes identifiées dans ses savoirs et savoir-faire durant le projet

5.1.1 Nicholas Legrand

La première lacune que j'ai rencontrée est le développement de logiciel adapté pour l'exécution sur un microcontrôleur, c'est-a-dire d'adapter les algorithmes pour qu'il s'exécute plus rapidement sur des microcontrôleurs comme le Arduino Uno. Je n'avais jamais eu besoin de développer une application avec des ressources si limitées.

La deuxième lacune que j'ai rencontrée est ma connaissance des mathématiques. Afin de faire une revue adéquate des implémentations des autres membres de l'équipe, je devais être en mesure de bien comprendre les algorithmes afin de comprendre les choix que mes coéquipiers ont effectués.

5.1.2 Gaya Mehenni

Dans le cadre de ce projet, j'ai identifié deux lacunes. Tout d'abord, j'ai éprouvé des difficultés à comprendre les différents algorithmes utilisés par les autres membres de l'équipe. En effet, certains de ces algorithmes m'étaient inconnus, ce qui a compliqué la revue du code écrit par mes pairs, notamment par rapport à l'algorithme de Méthode de Monte Carlo pour l'apprentissage par renforcement. En outre, j'ai également rencontré des difficultés liées aux implémentations matérielles. En général, je suis habitué à implémenter des programmes destinés à être exécutés sur des ordinateurs. Toutefois, les microcontrôleurs sur lesquels je travaillais avaient un processeur complètement différent de celui auquel je suis habitué, ce qui a nécessité une adaptation de ma part. Les restrictions de mémoire et les limitations de calculs à virgule flottante ont été particulièrement difficiles à surmonter, car je n'ai pas souvent eu l'occasion de programmer sur un microcontrôleur disposant d'aussi peu de mémoire.

5.1.3 Félix-Antoine Constantin

La première lacune que j'ai pu identifier est au niveau de ma compréhension des algorithmes implémentés par les autres membres de l'équipe. En effet, j'ai dû faire beaucoup de revues de code au cours de ce projet. Cependant, je ne possédais pas nécessairement toujours une bonne compréhension du fonctionnement de ces algorithmes.

La seconde lacune que j'ai identifiée est reliée à la compréhension de l'optimisation du code sur microcontrôleur. En effet, lors du projet de 1re année j'ai pu travailler avec des microcontrôleurs pour faire du code embarqué, mais je n'avais jamais touché à la partie optimisation. Il est important de bien comprendre le fonctionnement à l'interne d'un microcontrôleur pour optimiser le code et je ne possédais pas de bonnes connaissances à ce niveau au début du projet.

5.1.4 Alexis Foulon

La première lacune que j'ai identifiée durant le projet est mon manque de compréhension générale des éléments mathématiques des divers algorithmes que nous avions à implémenter. En effet, il était difficile pour moi de lire les preuves mathématiques expliquant le fonctionnement des algorithmes ce qui me causa beaucoup de difficultés lors de l'implémentation de ceux-ci.

Une seconde lacune que j'ai identifiée durant le projet est la difficulté de créer des programmes pour des architectures ou l'utilisation de mémoire est extrêmement restreinte. En effet, la consommation de mémoire n'a jamais été quelque chose dont je me préoccupe en programmation puisque nos ordinateurs ont tellement de mémoire vive disponible. Cependant avec un MCU ayant aussi peu que 2KB de mémoire vive ce fut une réalité à laquelle j'ai dû faire face.

5.1.5 Paul-André Bisson

La première lacune que j'ai pu observer de mon côté est la difficulté que j'avais à apporter des optimisations sur du matériel. Habituellement, j'apporte des optimisations à des programmes qui sont utilisés sur des ordinateurs. Dans le contexte de ce projet, les contraintes de mémoire étaient très importantes et les microcontrôleurs sur lesquels je travaillais avaient des modules de calculs entièrement différents de ceux auxquels nous sommes habitués sur un processeur classique. Toujours en lien avec l'optimisation, j'avais de la difficulté à identifier à quel point on désirait augmenter la performance au détriment de la précision. En effet, le client mettait de la connotation sur les optimisations de performance, mais il n'était pas toujours évident de déterminer à quel point je pouvais sacrifier de la précision pour augmenter celle-ci.

5.1.6 Florence Cloutier

Une première difficulté à laquelle j'ai été confrontée est celle de compréhension des différents algorithmes. Effectivement, le fonctionnement de plusieurs d'entre eux ne

m'était pas tout à fait connu. Cela a posé problème lors de l'implémentation et de la revue du code écrit par mes pairs.

Une seconde lacune observée est celle du développement et de l'optimisation de code pour l'exécution sur microcontrôleur. Étant habituée à programmer pour que le code roule sur un CPU, j'ai eu de la difficulté lorsque je devais prendre en compte les restrictions du microcontrôleur dans mon code. En effet, les restrictions de mémoire ont été le défi qui a été le plus difficile à surmonter du fait que j'ai rarement eu l'occasion de programmer sur un microcontrôleur avec autant peu de mémoire.

5.2 Méthodes prises pour y remédier

5.2.1 Nicholas Legrand

J'ai remédié ma première lacune qui était mon manque de connaissance du développement spécialisé pour les microcontrôleurs en faisant appel aux ressources qui m'étaient disponibles, c'est-à-dire que j'ai utilisé des ressources en lignes, les connaissances des membres de mon équipe ainsi que les connaissances de notre client. Le compilateur qui est utilisé pour compiler nos algorithmes sur Arduino Uno est avr-gcc. Il existe des solutions en ligne telles que Godbolt qui permette de voir le code assembleur généré par la compilation. J'ai pu voir la différence entre le temps d'exécution de différentes opérations et ensuite améliorer mon implémentation. Certains membres de notre équipe possédaient de l'expérience dans ce domaine et j'ai pu les consulter tout au long du projet pour avoir leur opinion sur différentes idées. Notre client possédait également quelques connaissances dans le milieu et nous a donné des conseils tout au long du projet.

La seule solution pour la deuxième lacune était de lire toute la documentation fournie par le client pour guider le développement des algorithmes. Après avoir fait cela, j'ai pu accomplir une revue complète de l'implémentation.

5.2.2 Gaya Mehenni

Pour remédier à ces lacunes, j'ai adopté plusieurs méthodes. Tout d'abord, j'ai consacré du temps à la lecture de la documentation des algorithmes utilisés dans le projet afin de mieux comprendre leur fonctionnement et leur implémentation par mes coéquipiers. J'ai également échangé avec mes collègues pour clarifier les points qui me semblaient encore flous. En ce qui concerne les implémentations matérielles, j'ai étudié attentivement les spécifications des microcontrôleurs sur lesquels je travaillais pour comprendre leur fonctionnement et les restrictions de mémoire auxquelles ils étaient soumis. J'ai ainsi pu adapter mes pratiques de programmation pour prendre en compte ces contraintes et optimiser mon code en conséquence.

5.2.3 Félix-Antoine Constantin

J'ai pu remédier à ma première lacune en posant des questions à mes collègues lors des revues de code. Lorsque je ne comprenais pas correctement le fonctionnement d'un algorithme, je pouvais demander conseil à la personne ayant écrit le code. Ceci m'a donc permis d'avoir une meilleure compréhension des algorithmes et d'effectuer des revues de code plus pertinentes.

J'ai pu remédier à ma seconde lacune en utilisant des outils tels que des compilateurs en ligne. Ces outils me permettent de visualiser le code assembleur associé à un code en C. Je peux donc plus facilement comprendre comment le code écrit est traduit pour le microcontrôleur et quelles sont ces faiblesses ou quels points peuvent être accélérés.

5.2.4 Alexis Foulon

J'ai pu remédier à ma première lacune en allant sur Youtube et en y regardant des vidéos expliquant les concepts mathématiques et les algorithmes que je ne comprenais pas. De cette manière il m'a par la suite été possible d'aller relire les descriptions mathématiques des algorithmes et d'être mieux aptes à les comprendre pour les implémenter.

J'ai pu remédier à ma seconde lacune en établissant des normes lors de mon codage. Notamment en m'efforçant de toujours utiliser le plus petit type ou la plus petite structure de mémoire permettant à mon algorithme d'accomplir son travail. De plus, j'ai aussi utilisé divers outils afin de voir la taille et l'utilisation de mémoire de mon programme me permettant de voir si des sections de celui-ci étaient trop gourmandes.

5.2.5 Paul-André Bisson

Pour remédier à mes difficultés d'optimiser efficacement les programmes développés pour des microcontrôleurs, il a fallu que je fasse plusieurs recherches sur le matériel visé. De plus, il a fallu que je me familiarise avec les fiches techniques des microcontrôleurs afin de mieux comprendre quelles optimisations sont plus efficaces que d'autres. En effet, puisque les modules de calculs sont totalement différents, certaines optimisations qui fonctionnent bien sur un processeur standard ne sont pas du tout efficaces sur certains microcontrôleurs. Dans ce cas, j'ai dû développer un module d'évaluation des performances afin de vérifier si les optimisations avaient l'impact désiré. Également, afin de mieux identifier les attentes quant aux optimisations voulues par le client, j'ai réalisé et évalué différentes versions optimisées des algorithmes. Je les ai alors présentées et j'ai pu obtenir sa rétroaction sur quelle version l'intéressait et à quel point il était prêt à sacrifier de la performance en faveur d'une accélération.

5.2.6 Florence Cloutier

Pour remédier à mon manque de savoir sur les différents algorithmes qui devaient être implémentés dans le cadre du projet, je me suis référée à beaucoup de documentation et de livres en ligne dans lesquels ces algorithmes et méthodes étaient décrits plus en

détail. Plusieurs des ressources auxquelles je me suis référée ont été proposées par notre client, qui lui-même est très renseigné sur le sujet. J'ai donc également bénéficié des connaissances du client pour pallier mon manque de connaissances sur les algorithmes.

Pour remédier à mon manque de savoir-faire en programmation pour microcontrôleurs, j'ai dû réfléchir plus longuement sur la structure de mon code, sur les structures de données utilisées et les types utilisés, pour rendre le code plus optimisé. J'ai également demandé de l'aide à mes coéquipiers, certains d'entre eux davantage avertis dans la programmation pour microcontrôleurs. J'ai également utilisé des outils comme Valgrind [9] et AddressSanitizer [10] pour mieux analyser mon code et déterminer comment l'optimiser davantage.

5.3 Identifier comment cet aspect aurait pu être amélioré

5.3.1 Nicholas Legrand

La première lacune aurait pu être remise en étudiant davantage la programmation pour microcontrôleur avant de commencer à développer l'algorithme. J'ai perdu du temps à optimiser l'algorithme une fois qu'il était déjà implémenté. J'aurais pu améliorer la deuxième lacune de la même manière, c'est-à-dire étudier tous les algorithmes dès le départ, ce qui m'aurait permis d'effectuer plus rapidement la revue des requêtes de fusion.

5.3.2 Gaya Mehenni

En ce qui concerne la première lacune, qui est la compréhension des algorithmes utilisés dans le projet, j'aurai pu planifier la revue de code et me familiariser avec les algorithmes avant la revue de code, afin de gagner du temps et de contribuer plus efficacement à la revue de code lors des requêtes de fusion.

En ce qui concerne les lacunes liées aux implémentations matérielles, j'aurai pu approfondir ma connaissance des spécifications des microcontrôleurs en lisant davantage de documentation et en effectuant des tests pratiques comme évaluer le nombre d'instructions assembleur générées par le compilateur en fonction des différentes implémentations de mes algorithmes. J'aurai également pu collaborer avec mes collègues qui peuvent être plus expérimentés dans ce domaine pour obtenir des conseils et des recommandations.

5.3.3 Félix-Antoine Constantin

Pour mieux améliorer la première lacune, j'aurais pu lire au préalable la documentation associée aux divers algorithmes. En effet, le client a fourni, pour chaque algorithme, plusieurs ressources en ligne permettant de nous familiariser avec celles-ci. Une lecture au préalable m'aurait permis de bien comprendre les algorithmes de tout le monde et donc d'être en mesure d'effectuer des revues de code plus efficaces.

Pour mieux améliorer la seconde lacune, une lecture de la documentation du microcontrôleur fourni aurait été pertinente. En effet, la compagnie fournit gratuitement plusieurs documents détaillant avec précision le fonctionnement des microcontrôleurs. J'aurais donc pu me référer à cette source dès le départ pour bien comprendre comment le code en C était transformé en assembleur et comment celui-ci fonctionnait par la suite sur le microcontrôleur.

5.3.4 Alexis Foulon

Pour mieux améliorer ma première lacune, je vais m'assurer dans le futur de toujours chercher à comprendre les bases mathématiques des algorithmes que j'applique en programmation. En effet, il me sera utile de me forcer à exercer mes compétences mathématiques sur une base hebdomadaire, que ce soit avec un manuel ou à l'aide de vidéo afin d'être sûr que je sois toujours à jour dans les compétences mathématiques nécessaires à la compréhension des algorithmes.

Pour ce qui est de ma seconde lacune, celle-ci aurait pu être améliorée en m'efforçant de toujours utiliser les structures de données qui consomment le moins grand nombre de ressources même dans les projets qui n'en ont pas forcément besoin. De cette manière, je serais toujours conscient des ressources utilisées par mes programmes.

5.3.5 Paul-André Bisson

Cet aspect aurait pu être grandement amélioré en communiquant plus souvent avec le client. En d'autres mots, puisqu'il existait de nombreuses incertitudes en lien avec le matériel sur lequel je devais travailler et qu'il était entièrement fourni par le client, il aurait été possible d'accélérer ma compréhension de ce dernier en posant des questions directement au client. De cette façon, il aurait été possible pour moi d'apporter de meilleures optimisations dans un délai plus court. Également, en ce qui concerne la gestion de la performance par rapport à la précision, communiquer plus souvent avec le client m'aurait permis de mieux gérer cette partie du projet. Pour ce qui a été développé, il a fallu que je passe beaucoup de temps à trouver des optimisations qui ne seront peut-être pas utilisées par le client. En ayant une relation plus étroite avec ce dernier, il aurait été possible d'éviter de passer trop de temps à optimiser, et d'utiliser ce temps pour améliorer la qualité des algorithmes sur lesquels j'ai travaillé.

5.3.6 Florence Cloutier

Afin d'améliorer mes connaissances sur les différents algorithmes et méthodes à implémenter dans la librairie, il aurait été pertinent de s'informer davantage. Un autre moyen pour approfondir mes connaissances aurait été de poser plus de questions au client, qui lui était très renseigné sur chacun des sujets. Il aurait également été pertinent de regarder des implémentations déjà existantes de ces algorithmes, dans d'autres langages que le C pour mieux visualiser comment ces algorithmes pourraient être définis.

Mon savoir-faire de développement d'algorithmes pour microcontrôleur aurait pu être amélioré si j'avais expérimenté davantage sur ceux-ci avant le projet. J'aurais également pu me référer à des exemples fournis dans la documentation Arduino pour mieux comprendre ses spécifications d'utilisation.

6. Références utilisées

[1] LLVM Project, The LLVM Compiler Infrastructure Project. [En ligne]. Disponible: https://llvm.org

[2] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, Second. Cambridge, USA: Cambridge University Press, 1992.

[3] Y. Saad, Numerical methods for large eigenvalue problems. Manchester, England:Society for Industrial and Applied Mathematics, 1992.

[4] "Line Search," dans Wikipédia, 20 juil. 2022. [En ligne]. Disponible : https://en.wikipedia.org/w/index.php?title=Line_search&oldid=1099362792

[5] Charbonneau, P. (2002). *An Introduction to Genetic Algorithms for Numerical Optimization* (No. NCAR/TN-450+IA). University Corporation for Atmospheric Research. doi:10.5065/D608638S

[6] A. Turing, ROUNDING-OFF ERRORS IN MATRIX PROCESSES, The Quarterly Journal of Mechanics and Applied Mathematics, Volume 1, Issue 1, 1948, Pages 287–308

[7] ClangFormat — Clang 17.0.0git documentation. (s. d.). [En ligne]. Disponible : https://clang.llvm.org/docs/ClangFormat.html

[8] GCC, the GNU Compiler Collection—GNU Project. (s. d.). [En ligne]. Disponible: https://gcc.gnu.org

[9] Valgrind Developers, Valgrind. [En ligne]. Disponible: https://valgrind.org

[10] The Clang Team, AddressSanitizer. [En ligne]. Disponible: https://clang.llvm.org/docs/AddressSanitizer.html

7. Annexe

Annexe 1 : Fichier de formatage .clang-format

```
---
Language:        Cpp
# BasedOnStyle:  LLVM
AlignAfterOpenBracket: Align
AlignOperands:   true
AlignTrailingComments: true
AllowAllArgumentsOnNextLine: true
AllowAllParametersOfDeclarationOnNextLine: true

AllowShortBlocksOnASingleLine: Never
AllowShortCaseLabelsOnASingleLine: false

AllowShortFunctionsOnASingleLine: All
AllowShortIfStatementsOnASingleLine: Never
AllowShortLoopsOnASingleLine: false
AlwaysBreakAfterReturnType: None
BraceWrapping:
  AfterCaseLabel:  false
  AfterControlStatement: false
  AfterEnum:       false
  AfterFunction:   false
  AfterStruct:     false
  AfterUnion:      false
  AfterExternBlock: false
  BeforeElse:      false
  IndentBraces:    false
  SplitEmptyFunction: true
  SplitEmptyRecord: true
BreakBeforeBinaryOperators: None
BreakBeforeBraces: Attach
BreakBeforeTernaryOperators: true
BreakStringLiterals: true
ColumnLimit:     80
ContinuationIndentWidth: 4
IncludeBlocks:   Merge
IndentCaseLabels: false
IndentWidth:     2
IndentWrappedFunctionNames: false
KeepEmptyLinesAtTheStartOfBlocks: true
```

```
MaxEmptyLinesToKeep: 1
PointerAlignment: Left
ReflowComments:  true
SortIncludes:    true
SpaceBeforeAssignmentOperators: true
SpaceBeforeParens: ControlStatements
SpaceInEmptyParentheses: false
SpacesBeforeTrailingComments: 1
SpacesInConditionalStatement: false
SpacesInContainerLiterals: false
SpacesInCStyleCastParentheses: false
SpacesInParentheses: false
SpacesInSquareBrackets: false
SpaceBeforeSquareBrackets: false
UseTab:          Never
…
```

Annexe 2 : Lien pour accéder au tag présent sur le répertoire Github du projet:

https://github.com/felix642/1chipML/tree/remiseFinale