
Rapport du Projet de Programmation
Pipeline pour l'analyse des maladies mitochondriales

UE Projet de Programmation
Master Bio-informatique
Université de Bordeaux
Mars 2023

Auteurs :

Esther Cros, Yaëlle Guiberteau, Mallory Le Corre,
Corentin Serain, Noah Vanney

Client :

Aurélien Trimouille
Centre Hospitalier Universitaire de Bordeaux

Encadrant :

Slim Karkar



CHU
Hôpitaux de
Bordeaux

université
de **BORDEAUX**

Table des matières

1	Introduction	3
2	Analyse	3
2.1	Contexte	3
2.2	Etat de l'art	3
2.3	Besoins et objectifs	4
2.3.1	Besoins fonctionnels	4
2.3.2	Besoins non fonctionnels	5
3	Conception	5
3.1	Etapes du pipeline	5
3.1.1	Alignement	5
3.1.2	Variant calling	6
3.1.3	Annotation	6
3.1.4	Informations complémentaires	8
3.1.5	Base de données	8
3.2	Organisation du pipeline	10
3.3	Outils de développement	10
3.3.1	Langages de programmation	10
3.3.2	Outils de travail collaboratif	11
3.4	Répartition des tâches	11
4	Réalisation	12
4.1	Alignement	12
4.2	Variant calling	15
4.3	Annotation	16
4.4	Informations complémentaires	18
4.4.1	Détection des réarrangements à l'aide d'eKLIPse	18
4.4.2	Identification de l'haplogroupe de l'individu par Haplogrep3	19
4.5	Conteneur Docker	21
4.5.1	Installations de Docker et du répertoire	22
4.5.2	Choix de l'image de base	22
4.5.3	Installation des logiciels	22
4.5.4	Mise en place du pipeline	23
4.6	Base de données	24
4.6.1	Création	24
4.6.2	Insertion	25
4.6.3	Requêtes	26
4.6.4	Utilisation	26
5	Conclusion	27
	Références	28

1 Introduction

Les mitochondries sont des organites cellulaires présents dans la plupart des cellules du corps. Elles sont responsables de la production d'énergie, nécessaire au fonctionnement de la cellule. Les maladies mitochondriales sont d'origine génétique[1]. Elles affectent la fonction mitochondriale, entraînant une baisse de la production d'énergie dans les cellules touchées[2].

Ces maladies touchent environ une personne sur 4000 dans la population. Les symptômes d'une maladie mitochondriale peuvent varier considérablement en fonction de la gravité de l'affection, de l'âge auquel elle commence[3], de la partie du corps qui est touchée et du taux d'hétéroplasmie. Les symptômes les plus courants incluent une fatigue généralisée, une faiblesse musculaire, une perte de coordination, des troubles du mouvement, des problèmes de vision, une perte auditive, des problèmes cardiaques et des troubles digestifs[4].

Il existe plusieurs types de maladies mitochondriales, chacune étant causée par une mutation génétique différente. Ces mutations peuvent être transmises de manière autosomique dominante ou récessive, ou bien être le résultat d'une nouvelle mutation génétique qui n'a pas été héritée. Il n'existe actuellement aucun traitement curatif pour les maladies mitochondriales. Les traitements visent seulement à atténuer les symptômes et à améliorer la qualité de vie des personnes atteintes de ces pathologies[5].

La fonction mitochondriale est en grande partie encodée dans l'ADN mitochondrial (ADNmt). Cet ADNmt est un chromosome circulaire d'une taille de 16 600 pb. Son séquençage est essentiel pour établir un diagnostic génétique précis et pour déterminer les risques de transmission. L'annotation des résultats de séquençage, qui inclut la position, le gène, la référence, le changement protéique ainsi que la présence dans des bases de données et la littérature scientifique, est essentielle pour interpréter les résultats.

Dans l'ensemble, le diagnostic génétique des maladies mitochondriales représente un défi important pour les professionnels de la santé. Il est aussi essentiel pour améliorer la prise en charge de ces maladies rares et pour fournir un conseil génétique adéquat aux patients et à leur famille.

2 Analyse

2.1 Contexte

Pour étudier les maladies mitochondriales, le CHU utilise actuellement différents outils et logiciels commerciaux qui ne sont pas mis à jour et qui sont donc peu performants. De plus, leur base de données est sous forme de fichiers Excel avec des macros, rendant l'étude des variants peu efficace. Ces méthodes ne sont pas à la hauteur du niveau d'exigence imposé par les normes de sécurité des données. Il est donc nécessaire de créer un pipeline optimisé et sécurisé pour ce type d'étude. Le pipeline est conçu de manière à être fonctionnel et facilement utilisable par tout membre du personnel autorisé et compatible avec les ordinateurs du CHU.

2.2 Etat de l'art

L'étude des maladies mitochondriales est en constante évolution. L'arrivée des NGS a permis une grande avancée dans la recherche en augmentant la précision et la rapidité du

séquençage[6].

En effet, en raison de la faible quantité d'ADN mitochondrial, une technique de séquençage ayant une haute fiabilité est indispensable. Un séquençage short reads est nécessaire car l'ADN mitochondrial est court. La technologie Ion Torrent répond à ces exigences en plus d'être rapide et efficace. Pour ces raisons, le CHU utilise actuellement un séquenceur Ion Torrent. Cependant, un passage à la technologie Illumina est en cours de réflexion.

Avant le séquençage, une PCR chevauchante ainsi qu'une migration sur gel sont effectuées.

L'analyse des données issues du séquençage est réalisée grâce à différents outils tels que BWA-mem, SAMtools ou GATK. Ces outils sont efficaces et offrent une grande gamme de fonctionnalités. Des outils informatiques tels que MITOMAP, VEP, eKLIPse et Haplogrep sont utilisés pour faciliter l'annotation et l'interprétation biologique.

2.3 Besoins et objectifs

2.3.1 Besoins fonctionnels

Afin de garantir une utilisation efficace et fiable du pipeline, il faut s'assurer que les résultats obtenus à chaque étape du pipeline soient cohérents et fiables. Il est donc important d'avoir une conception soignée et une mise en œuvre efficace des étapes suivantes : alignement par BWA-mem, détection de variant, annotation des résultats, association aux outils Haplogrep et eKLIPse et intégration à une base de données.

Ensuite, il a été important de mettre en place un environnement automatisé pour faciliter l'utilisation du pipeline. Cela comprend notamment la création de variables d'environnement pour automatiser les chemins d'accès et garantir une utilisation sans erreur.

Initialement, l'intention était de fournir des scripts d'installation pour tous les outils et dépendances nécessaires à l'exécution du pipeline, afin d'assurer la cohérence entre les différentes étapes. Cependant, afin de garantir une harmonisation efficace, une alternative a été envisagée, à savoir l'utilisation de Docker. Docker offre la possibilité de généraliser les versions des logiciels utilisés dans le pipeline. Cela permet d'assurer que des versions cohérentes de Perl, Python et autres dépendances sont utilisées pour toutes les étapes du processus. En utilisant Docker, il devient plus facile de maintenir la compatibilité et la reproductibilité du pipeline, en encapsulant toutes les dépendances logicielles dans des conteneurs indépendants et portables.

Il est également important de rendre les résultats facilement compréhensibles et utilisables par les professionnels de la santé chargés du diagnostic des maladies mitochondriales. Pour cela un output directement interprétable est nécessaire. Plusieurs fichiers sont accessibles à la sortie du pipeline. Un fichier BAM depuis l'étape d'alignement permet de créer via eKLIPse un csv ainsi que des graphiques, il permet également via le variant calling la production d'un vcf. Haplogrep à la suite sort un excel ou un fichier texte. Et enfin un fichier CSV de variants

annoté est consultable.

L'inclusion d'une gestion d'erreurs et d'exceptions a été faite et est primordial. En effet, signaler les problèmes de manière concise et claire peut faciliter le dépannage.

Enfin, le pipeline est modulaire et extensible, afin de permettre l'ajout de nouvelles fonctionnalités ou de nouveaux outils par le futur si besoin est.

2.3.2 Besoins non fonctionnels

Les besoins non-fonctionnel déterminent la facilité d'utilisation et la qualité globale d'un pipeline.

Premièrement, la clarté et la concision du code est importante. Il doit être facilement compréhensible pour les utilisateurs et pour les développeurs qui travaillent sur le pipeline. Pour cette raison, l'utilisation de commentaires a été privilégiée.

Deuxièmement, de la documentation permettra à l'utilisateur de mieux appréhender le pipeline. Dans cette dernière est spécifié le mode d'utilisation, les commandes de lancement, la méthode de modification des variables d'environnement, mais également, les options de commandes nécessaires, avec des exemples.

Le pipeline devant être compatible avec les outils et les technologies utilisés dans les laboratoires de diagnostic génétique. Il a été conçu pour s'intégrer facilement dans l'environnement existant et pour être utilisé avec d'autres outils selon les indications du clients.

Enfin, les noms de variables sont instinctifs et explicites afin de pouvoir remanier les scripts de manière efficace par le futur.

3 Conception

3.1 Etapes du pipeline

Le pipeline a pour objectif de fournir, à partir de la séquence d'ADNmt d'un patient, un maximum d'informations à un médecin habilité afin qu'il réalise un diagnostic le plus précis possible. Le pipeline sera composé de 5 étapes majeures :

3.1.1 Alignement

L'alignement sur un génome de référence est la première étape du pipeline. Cet alignement a pour input les reads issus du séquençage fournis par le client ainsi que le génome mitochondrial de référence qui a été obtenu sur le site NCBI, cette séquence de référence sera modifiée afin de supprimer les sauts de ligne entre les parties de la séquences, pour effectuer ces modifications deux commandes devront être utilisées, la première permettant de supprimer tous les sauts de ligne (`tr -d " < fichier.fasta > fichier_modifie.fasta`) et donc d'obtenir une référence sans saut de ligne, la deuxième commande permet d'effectuer un saut de ligne à la place 56 afin que l'entête et la séquence ne soit pas sur la même ligne.

```
sed -i 's/(\.{56}\.)/\1\n/' REF.fasta
```

L'alignement se fait ensuite grâce à deux outils : BWA et Samtools. BWA est un logiciel qui compare des séquences peu divergentes à un grand génome de référence. Plus précisément, il permet de faire l'indexation de la référence ainsi que l'alignement des reads avec cette dernière. Cet alignement se fait avec BWAmem qui est un algorithme de BWA. Le fichier obtenu en output est un fichier SAM. Samtools est ainsi utilisé afin de transformer le fichier SAM en fichier BAM. Cela afin de s'en servir pour le Variant Calling qui suit. Samtools va aussi servir pour autre chose car le génome étudié est mitochondrial et donc circulaire. l'utilisation de samtools dans l'algorithme bash permet de régulariser l'alignement sur ce génome circulaire.

L'outil Picard tools de MarkDuplicates est utilisé ensuite pour supprimer les résidus de PCR. En d'autres termes, il permet d'éliminer les reads identiques présents en plusieurs exemplaires car ils sont inutiles.

L'utilisation du logiciel IGV est également nécessaire. Il s'agit d'un logiciel permettant la visualisation de données génomiques. Dans le cadre de ce projet il permet de vérifier les différent alignement obtenu et va donc permettre de vérifier l'efficacité du code permettant cet alignement.

3.1.2 Variant calling

L'appel de variant consiste à identifier et caractériser les différentes variations de l'ADN mitochondrial. L'appel de variant prend en input le fichier BAM fourni par l'alignement ainsi que le fichier fasta du génome de référence. Ce fichier fasta doit être indexé pour que le script fonctionne.

Le variant calling se fait grâce au logiciel GATK qui est un outil d'analyse et de traitement de données de séquençage haut débit. L'outil Mutect2 inclu dans ce logiciel est utilisé car il propose de nombreux paramètres, dont un mode mitochondrial. Ce mode mitochondrial fixe automatiquement les paramètres adapté à un appel de variant sur un ADN mitochondrial. GATK utilisant Java en version 64 bits, il est donc nécessaire d'avoir la bonne version de Java ainsi que python en version 2 au préalable. [7]

Le fichier de séquençage aligné va subir une étape de variant calling qui va permettre la détection automatique des variants (variation génomique dans une séquence nucléotidique, en comparaison avec une séquence de référence).

Le fichier de sortie sera un fichier Variant Call Format (VCF). Ce fichier VCF représente les résultats de l'analyse génomique et contient une liste de variants détectés chez le ou les patients étudiés. Chaque ligne du fichier VCF représente un variant spécifique et comporte des informations telles que l'identifiant du génome de référence, la position de référence, l'identifiant du variant, les bases de référence, les bases altérées, la qualité, les filtres appliqués et d'autres informations additionnelles.

3.1.3 Annotation

L'annotation se fait à partir d'un fichier VCF fourni par l'étape précédente. L'objectif de cette étape est de déterminer si les variants détectés sont impliqués dans des maladies

mitochondriales connues ou non, ainsi que de prédire leur impact potentiel sur le patient. Pour atteindre cet objectif, une approche en deux étapes a été mise en place. Tout d'abord, une comparaison du fichier VCF à des bases de données déjà annotées a permis d'associer aux variants connus la maladie mitochondriale impliquée. Ensuite, une annotation via VEP prédit l'impact potentiel d'un variant inconnu sur les patients.

Annotation des variants connus avec Mitomap Mitomap est une base de données en ligne largement reconnue pour son rôle dans l'annotation des variants mitochondriaux. Il s'agit d'un recueil de polymorphismes et de mutations dans l'ADN mitochondrial humain[8]. Il a été choisi d'utiliser Mitomap en raison de sa fiabilité et de son exhaustivité dans la documentation des variants mitochondriaux associés aux maladies. Pour annoter les variants avec Mitomap, un script Python personnalisé a été développé. Il compare le fichier fourni par l'étape variant calling avec un fichier déjà annoté recherché sur Mitomap. Le script extrait les informations pertinentes sur les variants détectés, tels que l'identifiant du variant, la position de référence, les bases de référence et les bases altérées, et les associe aux maladies mitochondriales connues répertoriées dans Mitomap. Mitomap fournit des informations essentielles sur les variants connus, y compris leur fréquence dans la population, leur association avec des phénotypes spécifiques, et les références bibliographiques correspondantes. En intégrant ces informations au dataframe correspondant au fichier VCF, une annotation complète des variants déjà connus dans la littérature scientifique est obtenue.

Annotation des variants inconnus avec VEP L'annotation des variants inconnus est une tâche complexe car elle nécessite une prédiction de l'impact potentiel de ces variants sur le patient. Pour cela, VEP (Variant Effect Predictor) a été utilisé [9]. Cet outil est largement utilisé dans la communauté scientifique pour prédire les conséquences fonctionnelles des variants.

Selon McCarthy et al. (2014), dans leur article intitulé "Le choix des transcriptions et des logiciels a un effet important sur l'annotation des variantes"[10], le choix de l'outil d'annotation peut avoir un effet significatif sur les résultats obtenus. Comparé à ANNOVAR, un autre outil populaire pour l'annotation des variants, VEP offre plusieurs avantages significatifs. VEP est spécifiquement conçu pour l'annotation des variants et offre des fonctionnalités spécialisées dans la prédiction des conséquences fonctionnelles des variants. Il est également régulièrement mis à jour avec les dernières bases de données et annotations, ce qui garantit une annotation complète et à jour des variants.

VEP a été utilisé pour effectuer une annotation complète de tous les variants mitochondriaux trouvés chez les patients du CHU. Cette annotation a été réalisée en se basant sur une référence génomique spécifiée par le client.

En combinant les résultats des annotations de Mitomap pour les variants connus et de VEP pour les variants inconnus, un dataframe annoté complet a été fait, fournissant une vision globale des variants mitochondriaux identifiés chez le ou les patients étudiés. Afin d'être interprétée par des biologistes, la sortie devait être synthétique et facilement manipulable. Le choix a donc été fait de fournir un fichier CSV pour les étapes suivantes.

3.1.4 Informations complémentaires

Un haplogroupe correspond à une série de mutations présentes dans un chromosome. Il est donc détectable dans l'ADN d'un individu et peut être différent d'une population à l'autre, voir d'un individu à l'autre. Dans le cadre de cette étude sur l'analyse d'ADNmt, il est intéressant d'identifier ces haplogroupes. L'outil Haplogrep3[11] va en offrir l'opportunité. Haplogrep3 est un outil de classification d'haplogroupes rapide et gratuit qui prend en entrée un fichier d'ADNmt sous différents formats(VCF,FASTA,.txt) et qui donne en sortie les haplogroupes mitochondriaux de l'ADNmt, les variants annotés ainsi que des statistiques récapitulatives.

La détection précise des altérations de l'ADNmt est essentielle pour le diagnostic des maladies mitochondriales. Les techniques de détection sont en pleine progression mais il reste difficile de repérer les réarrangements multiples et en particulier les délétions multiples. Dans ce pipeline, l'outil qui permettra de répondre à cette problématique est eKLIPse[12]. Il s'agit d'un outil sensible et spécifique permettant la détection et la quantification de grands réarrangements d'ADNmt à partir de données de séquençage simples et appariées. Il contribuera à l'obtention des positions précises des points de rupture et du pourcentage cumulé de réarrangements de l'ADNmt au niveau d'un gène donné et cela avec une sensibilité de détection de mutant inférieure à 0,5%.

3.1.5 Base de données

L'idée est de construire une base de données regroupant tous les variants détectés chez des patients depuis 2015 ainsi que les variants détectés par le futur. Des annotations concernant ces variants seront aussi ajoutées à la base de données.

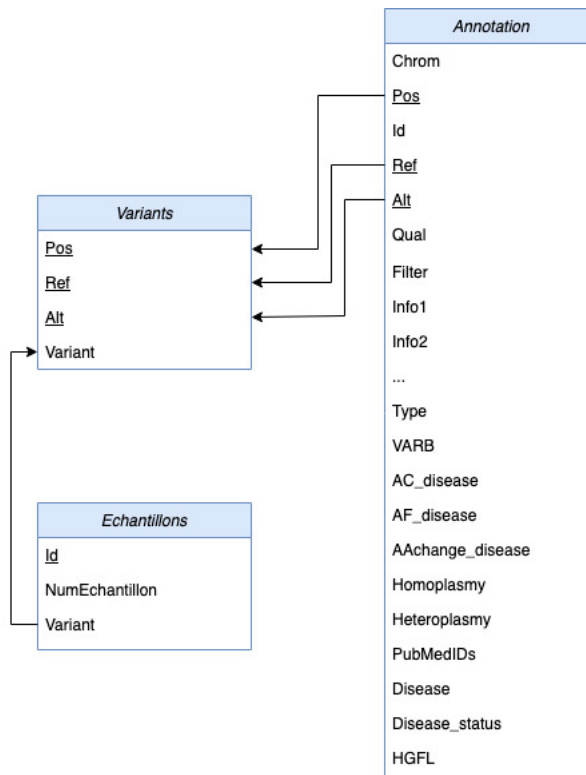


Figure 1 : Schéma de la base de données

La base de données se compose de trois tables. La table *Variants* contient les variants, selon la nomenclature utilisée au CHU, ainsi que la traduction de cette nomenclature en termes de position du variant, des bases de références et des bases altérées. La table *Echantillons* regroupe les numéros d'échantillons avec les variants détectés dans chaque échantillon. Afin d'identifier de manière unique chaque paire numéro d'échantillon-variant, on leur associe un identifiant (numéro) unique. La table *Annotation* regroupe les variants, identifiés par leur position, les bases de référence et altérées, ainsi que les annotations associées et contenues dans un fichier CSV produit à l'étape précédente (informations sur l'hétéroplasmie du variant, sur les maladies associées par exemple).

La clé primaire de la table *Variants* est la clé multiple (Pos, Ref, Alt). La clé primaire de la table *Echantillons* correspond à l'attribut Id. Enfin, la clé primaire de la table *Annotation* est la clé multiple (Pos, Ref, Alt).

Les tables *Echantillons* et *Annotation* ont toutes les deux des clés étrangères. La clé étrangère Variant de la table *Echantillons* fait référence à l'attribut Variant de la table *Variants*. La clé étrangère (Pos, Ref, Alt) de la table *Annotation* fait référence aux attributs Pos, Ref et Alt de la table *Variants*.

Les variants déjà séquencés sont regroupés dans des fichiers Excel. Il faut donc insérer les données depuis ces fichiers dans la base de données mais aussi créer une interface permettant l'insertion directe de nouvelles données sans passer par des fichiers Excel.

Il est judicieux de répartir le travail dans 4 fichiers : un premier pour créer l'architecture de la base de données sur un serveur de base de données, un second permettant d'insérer les données dans la base, un troisième afin de soumettre des requêtes SQL sur cette base de données et un dernier qui connectera les 3 fichiers précédents afin de rendre l'utilisation de la base de données plus pratique et intuitive pour l'utilisateur.

3.2 Organisation du pipeline

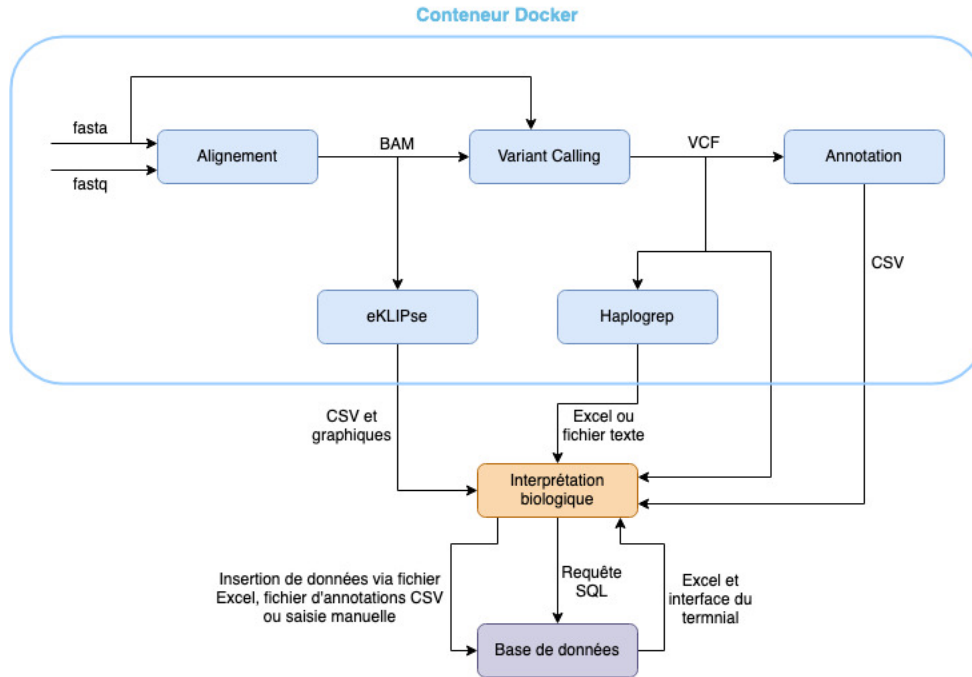


Figure 2 : Flowchart du pipeline

Les étapes d'alignement, de variant calling, d'annotation avec Mitomap et VEP ainsi que d'obtention d'informations complémentaires avec les outils Haplogrep et eKLIPse sont implémentées dans un conteneur Docker. Ce conteneur prend en entrée le fichier fastq des reads d'ADNmt du patient et fournit en sortie le fichier VCF issu du variant calling, le fichier CSV issu de l'annotation combinée avec Mitomap et VEP, un fichier CSV et des graphiques issus de l'annotation par eKLIPse et un fichier Excel ou .txt issu de l'annotation par Haplogrep.

Le biologiste peut ensuite choisir d'interagir ou non avec la base de données : en insérant de nouvelles données et/ou en soumettant des requêtes SQL sur les données.

3.3 Outils de développement

3.3.1 Langages de programmation

Bash Bash est un acronyme pour "Bourne Again SHell"[13]. Il est largement utilisé dans les systèmes d'exploitation basés sur Unix, tels que Linux et macOS. Le client a fait le vœux de faire un docker sous Linux. De ce fait il a été nécessaire d'en faire le langage principal. En effet, Bash est une application dont la fonction principale est d'exécuter d'autres applications installées sur un même système sous forme de commandes. L'avantage principal de cet outil est que le shell bash est "scriptable". C'est-à-dire que ce qui peut être entré manuellement sur Bash peut aussi être listé dans un fichier en texte plein et lancé par Bash.

Python Python est un langage de programmation qui permet de travailler rapidement et d'intégrer efficacement les systèmes [14]. Ce langage est employé pour GATK et eKLIPse en

version 2 et pour l'annotation et la base de données en version 3. Pour la base de données, les scripts en langage Python utilisent les librairies *pandas*, *sys* et *mariadb*.

Dockerfile Docker est une plateforme logicielle open-source permettant la création, le déploiement et l'exécution de conteneurs. Elle offre une solution pratique pour encapsuler des applications et leurs dépendances dans des environnements isolés et portables. Une des fonctionnalités clés de Docker est sa capacité à créer des images automatiquement en utilisant des fichiers Dockerfile[15]. Un Dockerfile est un document texte qui contient des instructions pour assembler une image Docker avec l'installation de packages ou de dépendances spécifiques, l'exécution de scripts et d'autres actions. Le processus de construction des images est basé sur des commandes exécutées via la ligne de commande, et il est spécifique aux systèmes Unix. Un avantage de Docker est sa compatibilité avec tous les systèmes d'exploitation. Les conteneurs Docker sont conçus pour être exécutés indépendamment du système d'exploitation employé par l'utilisateur. Souhaitant concevoir un pipeline cohérent, Docker nous a semblé être l'outil adéquat.

SQL Le "Structured Query Language" est un langage permettant de communiquer avec une base de données. La partie base de données utilise donc de nombreuses requêtes SQL.

Certains logiciels utilisent d'autres langages de programmation. On peut notamment citer Java et Perl.

3.3.2 Outils de travail collaboratif

GitHub GitHub[16], une plateforme de développement de logiciels adoptée par les développeurs, permet la collaboration, la gestion et le suivi des modifications apportées aux projets de programmation. Son utilisation a été décidée pour plusieurs raisons.

Tout d'abord, GitHub a permis de suivre l'avancée du projet de manière organisée. Chaque étape du processus a été répertoriée et suivie par tous les membres de l'équipe ainsi que par le client. Cette plateforme de développement a grandement facilité le travail d'équipe en fournissant un espace de partage de fichiers. Cela a été particulièrement utile pour les fichiers volumineux qui ont pu être transmis via cette plateforme.

De plus, un deuxième dépôt GitHub a été créé dans le but de fournir un rendu propre. Étant utilisé par Docker, il était nécessaire de créer un environnement bien organisé avec des scripts facilement accessibles. Cela afin de faciliter l'utilisation du conteneur.

Slack Slack est une application de messagerie pour les entreprises qui connecte les personnes aux informations dont elles ont besoin. [17] Les membres du groupes et le client ont ainsi pu être unifiés. C'est par ce biais que l'équipe à planifié les rencontres avec le client et posé des questions lorsque cela était nécessaire.

3.4 Répartition des tâches

Le pipeline étant composé de 5 étapes, il tombait sous le sens que chaque membre du groupe s'occupe d'une étape. L'idée n'était pas d'isoler le travail mais de l'optimiser afin que

la réalisation de chaque partie avance en parallèle. En effet, les fichiers sortant des uns étant les fichiers entrant des autres, l'organisation de réunions régulières a été nécessaire afin de faciliter la connection entre les étapes et ainsi rendre le pipeline fonctionnel.

Le travail a donc été réparti de la manière suivante : Corentin s'est occupé de l'étape d'alignement, Mallory s'est chargée du variant calling mais aussi de la mise en place du conteneur Docker, Esther a réalisé l'étape d'annotation, Noah s'est consacré à l'intégration des outils Haplogrep et eKLIPse au pipeline et Yaëlle s'est chargée de la base de données.

4 Réalisation

4.1 Alignement

L'alignement a été réalisé en plusieurs étapes en commençant tout d'abord par l'écriture d'un script bash qui prend les bons fichiers en input : la référence du génome mitochondrial humain au format FASTA ainsi que les reads séquencés au format FASTQ. Ce script permet d'obtenir un fichier BAM en output. Ce premier algorithme contient :

- Une vérification du bon nombre d'argument en entrée, dans le cas présent deux (référence et reads). L'utilisation d'un if permet de vérifier cela avec ["\$" -ne 2] qui vérifie si le nombre d'arguments est différent de deux, si c'est le cas l'algorithme doit s'arrêter et afficher ("Usage : \$0 reference_file.fastq reads_file.fastq") afin de préciser à l'utilisateur les inputs attendus. Sinon le script continue.
- La récupération du nom des fichiers en entrée. Cette étape permet à l'algorithme de donner un nom aux fichiers mis en entrée. Par exemple, reference="\$1" permet de dire que le premier fichier en entrée dans le script est le génome de référence sur lequel les reads seront alignés et donc que reads="\$2" est le deuxième fichier en entrée du script.
- La création d'un index pour le génome de référence grâce à BWA. La commande (bwa index "\$reference") va permettre cet indexation nécessaire au bon déroulement de l'alignement des reads par la suite.
- L'alignement des reads sur le génome de référence se fait grâce à BWAMEM[18]. BWA-MEM est l'un des trois principaux algorithmes de BWA. Il permet un alignement rapide et précis de reads de longueur comprise entre 70bp et 1Mbp. La commande utilisée est (bwa mem -t 4 "\$reference" "\$reads" > output.sam), l'option -t 4 utilisée avec BWAMEM signifie que l'alignement sera effectué en parallèle sur 4 threads. En d'autres termes, 4 processus se dérouleront en simultané. La suite de la ligne de commande est l'indication de la référence ainsi que des reads à aligner. Enfin, cet alignement est mis dans un fichier output.sam grâce au caractère >.

L'objectif de ce script est d'obtenir un .bam. Samtools a donc été utilisé afin de changer le fichier SAM en fichier BAM. Pour effectuer cette étape, la commande (samtools view -Sb output.sam > output.bam) a été réalisée. La commande samtools view est utilisée pour manipuler des fichiers au format SAM/BAM, qui contiennent des informations d'alignement génomique. -Sb sont des options de la commande samtools view : -S indique que le fichier d'entrée est au format SAM et -b spécifie que la sortie doit être convertie en format BAM (binaire SAM). Le fichier à traiter est ensuite ajouté à la commande puis > redirige la sortie

de la commande vers un nouveau fichier au format BAM.

Le fichier .bam est ensuite trié à l'aide de la commande (`samtools sort output.bam -o sorted.bam`). La commande `samtools sort` permet le tri du fichier `output.bam`. `-o` spécifie le nom du fichier de sortie trié qui est ici `sorted.bam`. Le tri des alignements selon leur position génomique permet de faciliter l'analyse des données mais surtout la lecture du fichier sur IGV.

Le fichier `sorted.bam` doit être ensuite indexé grâce à la commande `samtools index`. L'indexage du fichier BAM permet un accès rapide et efficace aux régions spécifiques du génome lors de l'analyse. La première étape a permis d'obtenir un alignement entre la référence du génome mitochondrial humain et les reads séquencés.

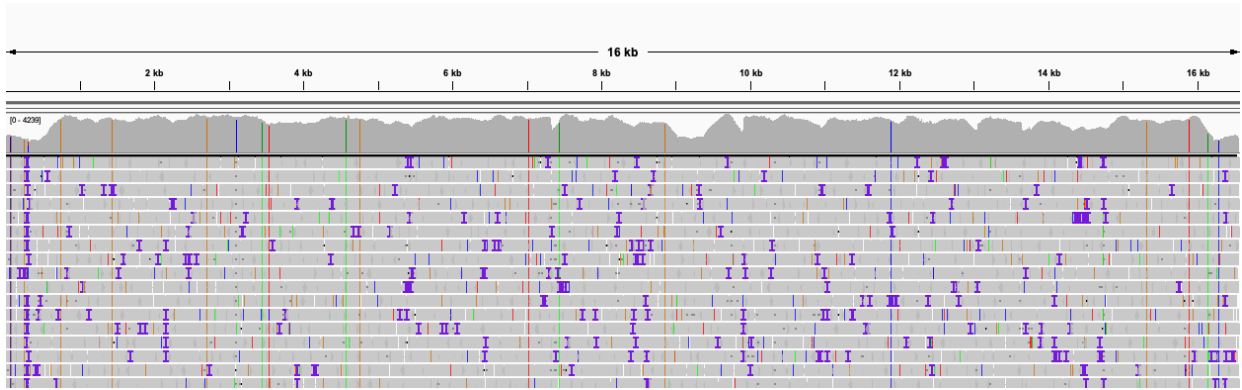


Figure 3 : Premier Alignement des reads sur la référence vue sur le logiciel IGV

Sur la figure, les résultats de ce premier algorithme sont exposés. Cependant, il est possible d'observer au début ainsi qu'à la fin une plus faible quantité des reads alignés. Cela s'explique par le fait que le premier algorithme ne prend pas en compte la circularité du génome mitochondrial. La solution pour régler ce problème est de copier les 300 premières bases de la référence pour les rajouter à la fin de celle-ci. L'utilité de cette manoeuvre est de permettre aux reads qui devraient s'aligner au niveau de la coupure du génome circulaire, c'est-à-dire ayant une partie qui doit être alignée au début de la séquence de référence et une autre partie sur la fin, de pouvoir s'aligner. Ensuite, il faut récupérer les bases alignées sur les 300 dernières bases de cette nouvelle référence (l'équivalent des 300 premières bases de la référence) et non les reads, puis les réaligner au début de cette référence.

Pour modifier le code il y aura donc plusieurs étapes à rajouter au script :

- Il faut tout d'abord réussir à copier les 300 premières bases de la référence pour les rajouter à la fin et donc créer une nouvelle référence. Prendre directement les 300 premiers caractères de la référence ne fonctionnerait pas car l'en-tête serait prise en compte. Il faut donc commencer par créer un fichier temporaire qui pourra être modifié, pour cela il faut utiliser la commande (`temp_file=$(mktemp)`).
- Ce nouveau fichier étant créé, il faut maintenant isoler les 300 premières bases de la référence d'origine. Il faut donc commencer par garder tout le fichier sauf l'en-tête (`grep -v ">" "$reference" | head -c 300 > "$temp_file"`). La commande `grep` permet cela. L'option `-v` permet d'inverser la recherche, c'est-à-dire de sélectionner toutes

les lignes qui ne correspondent pas au motif voulu. Dans ce cas, le motif est $\hat{>}$, qui recherche les lignes commençant par le symbole ">". Ainsi, toutes les lignes qui ne sont pas des en-têtes commençant par ">" seront sélectionnées. Ensuite, | permet d'effectuer une autre commande à la suite de la première, la commande head permet donc de récupérer les 300 premiers caractères sans l'en-tête qui ne fait plus partie de notre fichier. Le résultat de la commande est sorti dans le fichier temporaire.

- La référence et le fichier temporaire peuvent donc ensuite être fusionnés en un seul fichier grâce à la commande (cat "\$reference" "\$temp_file" > "\$reference_nouveau"), la commande cat permet au fichier temporaire d'être ajouté à la suite de la référence d'origine. Tout cela sera sorti dans un nouveau fichier. A la suite de cela, le fichier temporaire est supprimé grâce à la commande rm (rm "\$temp_file").
- La création d'un nouveau fichier FASTA est ensuite réalisé car il est nécessaire d'avoir un fichier FASTA pour réaliser l'alignement. echo "\$reference" > "nouveau.fasta") : la commande echo permet d'écrire et donc d'identifier dans le nouveau fichier FASTA en tant que référence. cat "\$reference_nouveau" » "nouveau.fasta" : la commande cat permet d'envoyer tout le fichier créé précédemment avec la référence longue de 300 bases supplémentaires dans le nouveau fichier FASTA.
- Après cette étape, il est important de constater qu'un saut de ligne se trouve entre la référence d'origine et les 300 bases ajoutées. Les mêmes manipulation que pour la référence issue de NCBI sont nécessaires et les lignes de commandes associées sont visibles en figure 3. Il y a deux sauts de ligne à ajouter cette fois car il y a, en plus de l'en-tête, l'identification en tant que référence du fichier FASTA.

```
tr -d '\n' < "nouveau.fasta" > "begin.fasta"
sed 's/(\.{9}\)/\1\n/' "begin.fasta" > "begin2.fasta"
sed 's/(\.{56}\)/\1\n/' "begin2.fasta" > "end.fasta"
```

Figure 4 : Premier Alignement des reads sur la référence vue sur le logiciel IGV

A la suite de ça il va falloir faire l'alignement comme vu précédemment puis récupérer les bases alignés sur les 300 dernières bases de la référence pour les aligner de nouveau au début de cette référence. (samtools view -h sorted.bam | awk -v ref="\$reference" 'BEGIN{OFS=" "}if(\$3==ref \$4 > (length(\$10)-300)){print}' | samtools view -bS - > "aligned_reads_end.bam") cette commande permet de récupérer les bases aligné dans le fichier BAM sur les 300 dernières bases de la référence.

La dernière partie consiste à supprimer les résidus de PCR afin d'enlever les reads en double pour cela Picard Tools de MarkDuplicates va être utiliser. Il est important de repréciser que l'utilisation de ce logiciel nécessite de détenir java. Les commandes de la figure permettent bien la suppression des duplicatats de PCR. Cette partie n'a pas encore pu être optimisé.

```
java -jar "$picard_jar" MarkDuplicates \
  I="$input_bam" \
  O="$output_bam" \
  M="$metrics_file" \
  REMOVE_DUPLICATES=true
```

Figure 5 : Suppression des duplicats de PCR

4.2 Variant calling

Pour le fonctionnement de gatk, python2 et java sont nécessaires, la première étape consiste donc en leurs installations. Cela peut poser des difficultés étant donné que restes des étapes du pipeline sont codées en python3.

L'outil prend en input le fichier bam ainsi que le fichier fasta de référence. Ces fichiers doivent être au bon format et une étape de pre-processing peut être nécessaire. Dans le cas de notre pipeline, le fichier fasta de référence sera déjà pré-traité avant l'étape d'alignement, il n'est donc pas nécessaire de le reproduire ici. (Il existe des outils pour vérifier le bon formatage de ces fichiers).

Le mitochondria-mode est un mode disponible dans Mutect2 qui installe tous les paramètres nécessaires à un appel de variante mitochondrial.

Il est aussi nécessaire de créer un index et un dictionnaire du fichier fasta. Pour l'index, l'outil samtools est utilisé avec la commande `faidx` suivit du fichier fasta. Cet index produit un fichier `ref.fasta.fai`. Une ligne est créée contenant le nom du chromosome, le nombre de bases total, la localisation, le nombre de bases et de bytes par lignes. Le dictionnaire est créé avec la commande `CreateSequenceDictionary -R` suivi du fichier fasta de référence. Le fichier contiendra une description de notre fichier fasta qui sera sous la forme d'un en-tête d'un fichier SAM. Sans ces deux fichiers, l'appel de variante ne fonctionnera pas. [7]

Un script a donc été créé contenant ces lignes de commandes. Il prend en input les données placées dans le fichier `datèrent`. Ensuite gatk est appelé, la ligne de commande sera sous la forme suivante :

```
gatk Mutect2 \
-R fastaReference.fasta \
-L chrM 1 \
--mitochondria-mode \
-I sorted.bam \
-O output.vcf.gz \
```

Figure 6 : Commande pour l'appel de gatk

Elle prend en input le fichier fasta de référence indexé et dont le dictionnaire a été créé ainsi que le fichier bam créé par l'alignement. Il faut aussi préciser le chrM. Mutect2 inclut un mode mitochondrial qui est passé en option. Ce mode mitochondrial permet d'imposer les paramètres suivants : le paramètre initial-tumor est paramétré à 0, cela permet d'inclure les variants même quand ils ont une faible vraisemblance. Le `tumor_lod` lui aussi paramétré à 0, ainsi tous les variants seront inclus dans les fichiers vcf. Le `-af-of-alleles-not-in-resource` est réglé sur le 4e-3, cela signifie que la fréquence allélique choisie comme fréquence par défaut est fixé à 0,004 ce qui est relativement bas. Le dernier paramètre est `-pruning-lod-threshold` qui est réglé sur -4, ce qui indique le seuil de suppression des variants qui se répètent.

Le script a été testé avec les fichiers test `Ion_Xpress.bam` et le fasta de référence. Cependant, le fichier `.bam` n'étant pas au bon format, l'appel de variant n'a pas pu être effectué sur les fichiers tests.

4.3 Annotation

Pour accomplir la tâche d'annotation, l'idée à tout d'abord été de partir sur des listes contenant les informations de chaque fichiers à mettre en parallèle. Les listes offrent une structure simple pour stocker et manipuler des données, tandis que `with open` permet de gérer l'ouverture et la fermeture de fichiers de manière explicite. Il s'agit de deux concepts fondamentaux de la programmation faciles à comprendre. Cependant, lorsque le code se complexifie, on se rend compte que travailler avec la bibliothèque `pandas` offre de nombreux avantages, tels que des fonctionnalités plus avancées pour la manipulation de données tabulaires et une gestion simplifiée des opérations courantes. Sachant cela, un remaniement du code a été fait.

Ainsi il a tout d'abord fallu importer les bibliothèques nécessaires pour effectuer les opérations requises. Dans ce cas, il est nécessaire d'avoir la bibliothèque `pandas` pour manipuler les données tabulaires, la bibliothèque `subprocess` pour exécuter des commandes shell, et la bibliothèque `os` pour effectuer des opérations sur le système d'exploitation. La première phase du script d'annotation est l'annotation par VEP pour les variants inconnus. La fonction `annotate_vcf(vcf_file)` prend en entrée un chemin de fichier VCF et a pour but d'annoter le fichier VCF à l'aide de la base de données Variant Effect Predictor (VEP). Elle utilise la bibliothèque `subprocess` pour exécuter une commande shell qui effectue l'annotation. La fonction vérifie également si le fichier de sortie annoté a été créé avec succès.

Après cela un appel à la fonction `load_and_parse_vcf(file_path)` est fait à 3 reprises, pour charger en dataframe les trois fichiers à traiter. Cette fonction prend en entrée un chemin de fichier VCF. Elle utilise la bibliothèque `pandas` pour lire le fichier ligne par ligne, extraire les en-têtes et les données des lignes du fichier, et créer un dataframe à partir de ces informations. Un traitement particulier est fait pour la colonne "INFO" qui contient des informations devant être séparé.

Les données des fichiers `"TSVC_variants_IonXpress_011.vcf"` (fichier test censé être obtenu par l'étape de variant calling) et les données du fichier `"disease.vcf"` (recherché sur Mitomap) sont fusionnées afin de ne conserver dans un nouveau dataframe que les informations pour les variants communs aux deux documents. La fonction `"merge"` de la bibliothèque `Pandas` permet cela avec l'option `"inner"`. Ce nouveau dataframe porte le nom de `"Merged_data_Mitomap"`. Il contient les informations sur les variants connus dans la littérature, les maladies auxquelles ils sont associés et les publications dans lesquels on peut les retrouver.

Subséquentement, le code fusionne le dataframe `"merged_data_Mitomap"` et le dataframe `"csv_variant_vep"` (dataframe résultant de la fonction `annotate_vcf`) en utilisant les colonnes `'CHROM'`, `'POS'`, `'REF'` et `'ALT'` comme clés d'index. Les valeurs manquantes dans `"mer-`

ged_data_Mitomap" sont remplacées par les valeurs correspondantes de "csv_variant_vep". Notez que les colonnes 'CHROM', 'POS', 'REF' et 'ALT' doivent être présentes dans les deux dataframes et avoir les mêmes noms et types de données pour que cette opération puisse être effectuée correctement. Le dataframe résultant est ensuite réindexé et possède l'intégralité des variants annotés, par la méthode Mitomap en priorité et par VEP en complément.

Un traitement du dataframe est ensuite nécessaire. Premièrement, le code convertit toutes les valeurs de la colonne 'CSQ' du dataframe "merged_data_anno" en chaînes de caractères, et remplit les éventuelles valeurs manquantes avec une chaîne vide. Cela permet de s'assurer que la colonne 'CSQ' ne contient que des chaînes de caractères et facilite les opérations ultérieures qui nécessitent des manipulations de chaînes de caractères. Deuxièmement, l'appel de la fonction `separate_csq_column(df, column_name)` prend en entrée le dataframe et le nom d'une colonne à séparer avec le séparateur. Son objectif est de séparer les éléments de la colonne donnée pour chaque ligne du dataframe en utilisant le délimiteur spécifique "|". Ensuite, elle crée de nouvelles colonnes pour chaque élément séparé et renomme ces nouvelles colonnes avec des noms prédéfinis. Enfin, une approche consistant à traiter les colonnes vides aurait amélioré le résultat, mais pour des raisons d'intégration ultérieure dans une base de données, il est nécessaire d'avoir des colonnes fixes.

Le code offre enfin la possibilité d'organiser les colonnes à la convenance du biologiste. Pour cela quelques lignes de codes suffisent. Tout d'abord, une liste appelée `ordering_columns` qui contient les noms des colonnes dans l'ordre souhaité est créée, ensuite, les colonnes restantes du dataframe y sont ajoutées. Enfin par la méthode `reindex()` de pandas les colonnes du DataFrame `merged_data_anno` sont réorganisé.

Le DataFrame réorganisé est ensuite exporté vers un fichier CSV avec le chemin d'accès spécifié dans `output_file`.

CHROM	POS	ID	REF	ALT	QUAL	TYPE	Disease	DiseaseStatus	FILTER	FORMAT
M	11899.		T	C	34544.3	snp			PASS	GT:GQ:DP:f
M	1438.		A	G	34511.8	snp			PASS	GT:GQ:DP:f
M	15326.		A	G	34642.1	snp			PASS	GT:GQ:DP:f
M	15904.		C	T	34655	snp			PASS	GT:GQ:DP:f
M	16153.		G	A	34556.5	snp			PASS	GT:GQ:DP:f
M	16298.		T	C	23982.8	snp			PASS	GT:GQ:DP:f
M	263.		A	G	25201.2	snp			PASS	GT:GQ:DP:f
M	2706.		A	G	34608.5	snp			PASS	GT:GQ:DP:f
M	309.		CTCCCCCG	CTCCCCCG.CTCCCCCG.CTCCCCCG.CTCCCCCG.CTCCCCCG	13302.4	snp.ins.ins.mnp			PASS	GT:GQ:DP:f
M	3460.		G	A	34217.2	snp	LHON	Cftr	PASS	GT:GQ:DP:f
M	3549.		C	T	34279	snp			PASS	GT:GQ:DP:f
M	4580.		G	A	15800.3	snp			PASS	GT:GQ:DP:f
M	4769.		A	G	20370.2	snp			PASS	GT:GQ:DP:f
M	7028.		C	T	33753.5	snp			PASS	GT:GQ:DP:f
M	72.		T	C	14778.4	snp			PASS	GT:GQ:DP:f
M	7444.		G	A	34026.2	snp	LHON-/SNHL-/DEAF-modulator	Reported	PASS	GT:GQ:DP:f
M	750.		A	G	34415.2	snp			PASS	GT:GQ:DP:f
M	8860.		A	G	7415.46	snp			PASS	GT:GQ:DP:f

Figure 7 : Première partie du fichier CSV produit par l'étape d'annotation

Avec le fichier test fourni, le script de cette étape rend une annotation utilisable et compréhensible par le biologiste. Toutes les informations présentes sur les VCF initiaux sont à ce jour mis à disposition dans le fichier CSV. Un tri des données pourrait être envisageable afin d'offrir un accès plus sélectif aux informations, cependant seule une discussion avec les biologistes concernés pourrait permettre cet épissage.

4.4 Informations complémentaires

4.4.1 Détection des réarrangements à l'aide d'eKLIPse

Le bon fonctionnement d'eKLIPse[12] nécessite l'installation des modules suivants : python 2.7, biopython, tqdm, samtools, blastn et makeblastdb (version supérieure à 2.3.0). Biopython va permettre la manipulation de séquences nucléotidiques et samtools l'interaction avec les fichiers au format BAM. Pour ce qui est de blastn et makeblastdb, ils permettent de comparer une séquence requête avec une base de données de séquences pour trouver des similitudes.

Une fois ces modules installés, l'installation du logiciel est possible à partir d'une archive présente sur le GitHub. Il suffit ensuite de unzipper cette archive et tous les scripts permettant le fonctionnement d'eKLIPse sont désormais disponibles dans le répertoire. C'est évidemment eKLIPse.py qui va permettre l'utilisation du logiciel. L'exécution d'eKLIPSE est réalisée par la commande suivante :

```
python eKLIPse.py -in <INPUT file path> -ref <GBK file path> [OPTIONS]
```

Le paramètre -in prend en argument le fichier BAM obtenu lors de l'étape d'alignement du pipeline et le paramètre -ref prend en argument un fichier Genbank de référence. Dans ce cas, les patients étant humains, le fichier de référence sera le génome mitochondrial complet de Homo Sapiens (NC_012920).

Les fichiers sortants sont au nombre de 3 dont 2 fichiers Excel renseignant sur les délétions et les duplications ainsi qu'un plot récapitulatif de l'analyse.

Pour des raisons inconnues, il semble qu'il y ait des problèmes liés à l'encodage des fichiers BAM qui empêchent le programme de fonctionner. Néanmoins, l'équipe est parvenue à faire tourner le programme test qui permet d'obtenir les 2 fichiers Excel. Malheureusement, il subsiste une erreur dans la réalisation des plots liée à la bibliothèque circos qui ne fonctionne pas.

Ayant réussi à se procurer une version portable d'eKLIPse avec une interface graphique, les membres de l'équipe ont pu obtenir les résultats escomptés. Ci-dessous, le graphique circos obtenu avec le fichier BAM test obtenu après l'étape d'alignement :

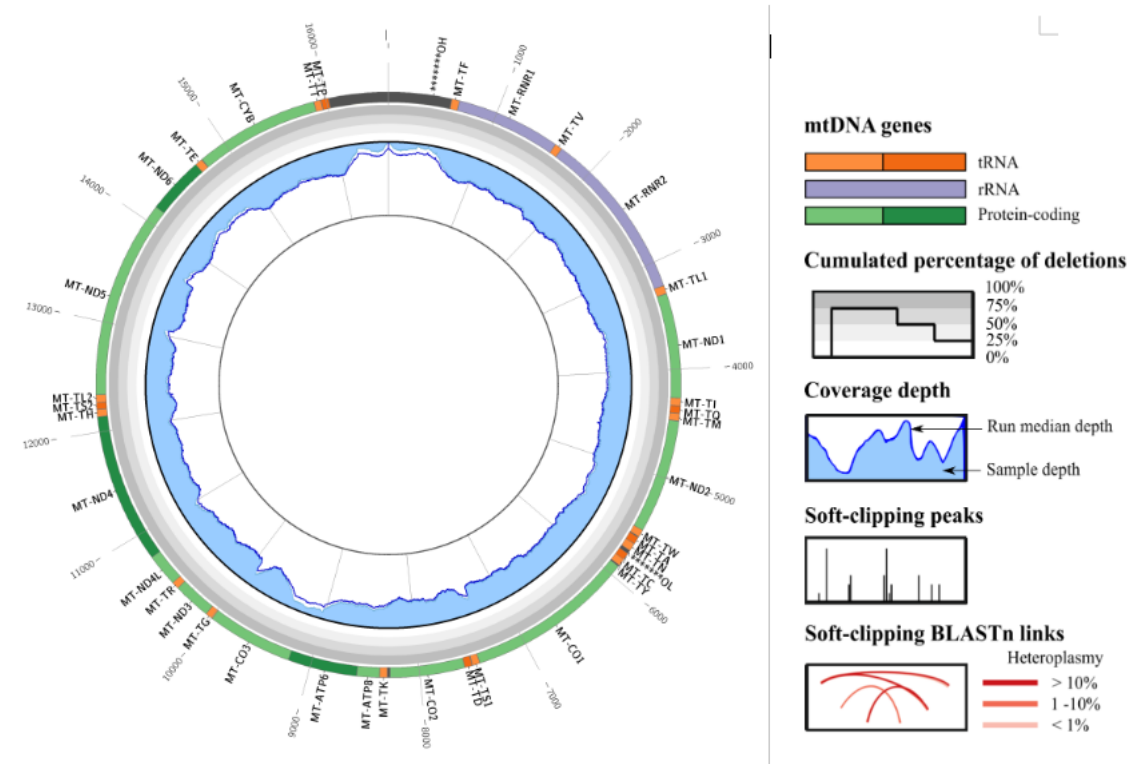


Figure 8 : Circos plot

Ce graphique est très complet et contient énormément de renseignement sur l'alignement. Tout d'abord, il décrit quels gènes sont présents dans l'ADN mitochondriale. Cette information est non négligeable quand il est assuré que l'ADN mitochondrial contient son propre ensemble de gènes distincts de ceux présents dans le noyau de la cellule.

De plus, il va donner des informations sur le pourcentage cumulé de délétions de l'ADN mitochondriale. C'est intéressant notamment pour évaluer l'intégrité totale de l'ADN mitochondriale.

Dans la partie interieur du cercle, la courbe circulaire bleu mesure la quantité de reads couvrant chaque position spécifique de l'ADN mitochondriale, elle est aussi nommé Coverage Depth. Une couverture élevée indique qu'une position spécifique a été lue à plusieurs reprises et renforce donc la qualité des données obtenues.

Les pics de soft-clipping indiquent les régions où la séquence lue correspond le moins à la séquence de référence.

Enfin, Soft-clipping BLASTn links indiquent les régions où les séquences de soft-clipping correspondent à des séquences ressemblant à celles présentes dans la base de données Blastn.

4.4.2 Identification de l'haplogroupe de l'individu par Haplogrep3

L'installation du logiciel Haplogrep[11] se réalise à partir d'un GitHub et nécessite au préalable de détenir une version de Java 8 ou supérieure. Une fois ces prérequis obtenus,

il suffit d'utiliser la commande `wget` pour aller chercher l'archive sur le GitHub. Le fichier archive sera ensuite unzipé et Haplogrep sera alors exécutable.

```
noah@DESKTOP-997JIIG:~$ wget https://github.com/genepi/haplogrep3/releases/download/v3.2.1/haplogrep3-3.2.1-linux.zip
```

Haplogrep peut prendre différentes options mais dans le cadre de cette étude, c'est l'option "classify" qui est intéressante. Cette option va permettre de classifier l'haplogroupe d'une séquence d'ADN mitochondrial. L'option "classify" va prendre 3 arguments obligatoires pour son fonctionnement :


- Un fichier d'entrée qui est, ici, le fichier au format VCF obtenu après l'étape de variant calling décrite précédemment
- Un arbre phylogénétique qui décrit la phylogénie mitochondriale. Ici, c'est l'arbre "PhyloTree 17 - Forensic Update" qui a été sélectionné puisqu'il s'agit de l'arbre le plus à jour pré-enregistré dans Haplogrep
- Un fichier de sortie avec son emplacement. Le fichier aura l'extension .txt pour faciliter les étapes ultérieures.

Enfin, un dernier paramètre optionnel sera ajouté : `--write-qc`. Il va induire la création d'un fichier .csv supplémentaire comportant un contrôle qualité sur les résultats.

```
noah@DESKTOP-997JIIG:~$ ./haplogrep3 classify --in=TSVC_variants_IonXpress_011.vcf. --tree=phylo tree-fu-rcrs@1.2 --out=haplogroup.txt --write-qc
```

Note : Les fichiers ci-dessous sont obtenus après exécution d'Haplogrep sur le fichier VCF test obtenu après l'étape de variant calling.

Le premier fichier obtenu en sortie est le fichier `haplogroup.txt`. C'est un fichier au format .csv composé de 5 colonnes délivrant des informations sur l'haplogroupe de l'individu.



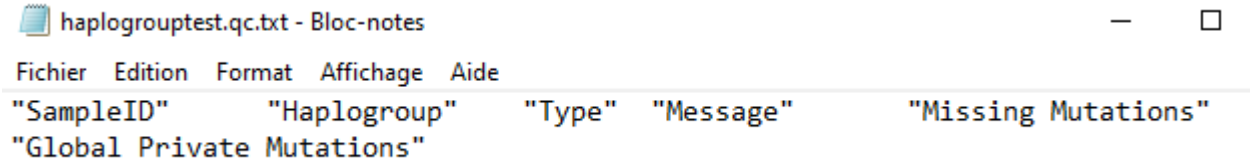
SampleID	Haplogroup	Rank	Quality	Range
J_N2300062	V7a1	1	1.0000	1-16569

Figure 9 : Fichier Haplogroupe Test

- La première colonne "Sample_ID" contient l'identifiant de l'échantillon étudié.
- La seconde colonne "Haplogroup" contient l'haplogroupe mitochondrial attribué à l'échantillon étudié.
- La troisième colonne "Rank" fournit un classement de l'haplogroupe attribué à l'échantillon. Les haplogroupes sont organisés en une hiérarchie, où certains haplogroupes sont plus spécifiques ou plus précis que d'autres. Le rang indique à quel niveau de la hiérarchie l'haplogroupe attribué se situe. Un rang plus élevé indique un haplogroupe plus spécifique.
- La quatrième colonne "Quality" représente une mesure de la qualité de l'assignation de l'haplogroupe. Il s'agit d'un score numérique qui évalue la confiance de l'attribution de l'haplogroupe basée sur les caractéristiques génétiques de l'échantillon. Plus la qualité est élevée, plus l'assignation est dite "fiable".

- La dernière colonne "Range" fournit une plage de positions sur lesquelles repose l'assignation de l'haplogroupe. Elle permet de comprendre les caractéristiques spécifiques qui ont été prises en compte dans la détermination de l'haplogroupe mitochondrial de l'échantillon.

Le deuxième fichier obtenu en sortie induit par l'option `-writeqc` est `haplogrouptest.qc`.



"SampleID"	"Haplogroup"	"Type"	"Message"	"Missing Mutations"
"Global Private Mutations"				

Figure 10 : Fichier de contrôle qualité Haplogroupe Test

- Les deux premières colonnes sont les mêmes que pour le fichier `haplogroup.txt` classique. Elles contiennent respectivement l'identifiant de l'échantillon et l'haplogroupe mitochondrial qui lui a été attribué.
- Les colonnes "Type" et "Message" indiquent des remarques ou des avertissements sur les résultats de qualité obtenus.
- La cinquième colonne "Missing Mutations" indique le nombre de mutations manquantes dans l'échantillon par rapport à une référence ou à un haplogroupe donné. Ces mutations peuvent indiquer des incohérences ou des variations par rapport à l'haplogroupe attendu.
- Enfin, la dernière colonne "Global Private Mutations" est la plus intéressante dans notre perspective d'analyse de maladies mitochondriales. Cette colonne recense des mutations spécifiques qui sont uniques à cet échantillon et qui ne se trouvent donc pas dans les haplogroupes connus ou de référence. Ce sont ces mutations que le biologiste devra regarder en priorité.

Dans un souci de confort d'analyse pour le biologiste, les deux fichiers de sortie sont fusionnés et transformés en fichier Excel à l'aide d'un script Python et plus particulièrement de la librairie `pandas`. Ce script prend en compte le fait que les deux fichiers ont deux colonnes identiques ("Sample_ID" et "Haplogroup") et affiche ainsi seulement les colonnes uniques. Ainsi, le biologiste aura accès à toutes les informations dans un seul fichier et sera plus apte à tirer des conclusions précises à partir des données.

4.5 Conteneur Docker

Dû à la différence de compatibilité entre les logiciels et les différentes machines, il a été décidé d'utiliser un Docker. Le `dockerfile` est écrit dans un langage propre à l'écriture d'un `dockerfile`. La difficulté du docker est le manque d'interaction lors de la création du `dockerfile`. Il est difficile de savoir dans quels répertoire chaque installation est faite et comment le container sera conçu. Aussi, alors qu'il est facile de constater la bonne installation d'un outil depuis le terminal d'un ordinateur, elle n'est pas vérifiable depuis le `dockerfile`. C'est pour cela que la ligne de commande `docker exec -it <id-du-docker> /bin/bash` a beaucoup

été utilisée. Elle permet en effet de passer en mode interactif en langage bash. Il est ainsi possible d'exécuter des commandes bash et cela permet une meilleure gestion des chemins d'accès et de faciliter la détection et la résolutions des erreurs.

4.5.1 Installations de Docker et du répertoire

La première étape consiste donc en l'installation de Docker. Cette installation diffère suivant le processeur et l'OS des différentes machines. Une fois installé, il faut créer le répertoire dans lequel le docker sera lancé. Dans ce répertoire sera créé dans un premier temps un fichier.sh. Ce fichier.sh contiendra un script contenant les lignes nécessaires pour lancer le docker. La première est la construction de l'image. *image du script* Le nom de cette image est définie à ce moment, et sera appelé *nom de l'image* ?? La partie située à droite des deux points constitue le tag, c'est à dire les différentes versions de l'image. La deuxième ligne de ce script sert à la construction d'un container. Ce container est créé à partir de l'image précédemment créé. Un conteneur est identifié grâce à son nom et à son ID. L'ID est récupéré dans la ligne suivante et placé dans une variable. Cette variable est utilisé dans la commande qui suit et qui sert à récupérer le dossier "output". Dans ce dernier se trouverons tous les outputs créé au cours du pipeline. Dans le même répertoire, le dockerfile sera placé. C'est à partir de ce dockerfile que sera créé l'image. Le dockerfile doit être nommé Dockerfile, sans extension et avec le D en capitale, sans quoi il faudra préciser dans la ligne de commande le nom du dockerfile. Dans ce répertoire devront aussi être placé les différents inputs nécessaire aux différentes étapes. [15]

4.5.2 Choix de l'image de base

La première ligne définit l'image de base utilisé. Il est important de bien choisir son image de base, car elle définira les packages déjà utilisé. Il existe des images très pratiques pour une application en NGS, comme c'est le cas pour biocontainer. C'est cependant une image ubuntu qui a été choisie, car c'est une image connue, contenant déjà beaucoup de packages et de logiciels utilisés dans le pipeline, comme c'est le cas de bash qui sera utilisé pour lancer toute les lignes de commandes ainsi que apt pour l'installations. Certains outils ne sont pas directement utilisés dans le pipeline mais ont été très utilisés dans l'installation du dockerfile. Ils peuvent aussi être utile si l'utilisateur veut modifier directement le conteneur. Il est important de préciser quelle version de ubuntu est exploité. Dans le cas de ce docker, c'est la version la plus récente qui est utilisé. Il est possible de préciser "latest" comme version pour avoir toujours la plus récente. Cependant il est important de spécifier une version qui reste inchangé. En effet, utiliser le tag "latest" est susceptible d'entraîner des variations à chaque nouvelle mise à jours, et donc entrainer des changements dans la compatibilité. [?]

4.5.3 Installation des logiciels

Tout d'abord, différentes dépendances utiles aux installations futures sont implémentées. Certaines sont des bibliothèques qui servent au bon fonctionnement du systèmes. Elles peuvent proposer différents types de fonctionnalités : gestion d'interfaces, affichage de textes,

compressions et autres. Les plus importantes dans ce lot sont perl, un langage de programmation utilisé dans plusieurs des logiciels qui seront utilisés plus tard, wget qui permet d'obtenir des fichiers depuis internet ou encore make qui automatise la construction de fichiers. Bwa étant déjà pré-installé sur l'image ubuntu, l'installation est plus simple. Pour le reste des logiciels et outils, les étapes sont globalement les mêmes. La commande wget est utilisée pour aller chercher les fichiers jar sur leur github puis déplacé dans le bon répertoire du dockerfile, les droits sont changés afin de les exécuter et le chemin d'accès est placé dans le PATH.

Pour les langages de programmation java et python, il est important de les actualiser aux versions nécessaires aux logiciels venant à la suite. Par exemple, gatk et eklipse sont codés en python2, tandis que le reste ainsi que les scripts utilisés pour le pipeline ont été programmé en python 3. Il est donc important, avant chaque étape, de mettre le langage à la bonne version.

Enfin, un des gros problème rencontré à été l'installation de VEP. Cet outil nécessite l'installation du génome de référence humain qui a un poid de 25 GO. Le volume n'étant pas supportable par toutes les machines personnelles des membres de l'équipe il a fallut faire les tests avec un génome plus petit (celui d'un poisson).

```
##### INSTALLATION DE PICARD TOOLS #####
# Picard Tools
RUN wget https://github.com/broadinstitute/picard/releases/download/2.25.6/picard.jar

RUN mv picard.jar /usr/local/bin/picard.jar
ENV PICARD_HOME /usr/local/bin

RUN echo '#!/bin/bash' >> /usr/local/bin/picard
RUN echo 'java -jar /usr/local/bin/picard.jar "$@"' >> /usr/local/bin/picard
RUN chmod +x /usr/local/bin/picard

# ajout du chemin d'accès dans le PATH
ENV PATH $PATH:/usr/local/bin
```

Figure 11 : Exemple des différentes étapes de l'installation d'un logiciel dans le Dockerfile avec Picard tools

4.5.4 Mise en place du pipeline

Un répertoire data est créé dans le conteneur. Grâce à la commande COPY de dockerfile, les fichiers utilisés dans le pipeline y sont transférés. Il est important d'implémenter les chemins aux fichiers dans les différents scripts employés.

```
# Clone du référentiel GitHub, copie de son contenu dans le repertoire "github"
RUN git clone https://github.com/yguiberteau/Pipeline-pour-l-analyse-des-maladies-mito /tmp/repo && \
cd /tmp/repo && \
# Un pull est effectuée afin d'avoir toujours la dernière version du GitHub
git pull && \
cp -r /tmp/repo/* /github && \
rm -rf /tmp/repo

RUN mkdir -p /.github/workflows
```

Figure 12 : Clonage du github dans le dockerfile

Le répertoire output est aussi créé. C'est dans ce répertoire que seront placés les différents fichiers obtenus en sortie au cours des étapes. De la même façon que pour le répertoire

data, il est important de s'assurer que ces chemins d'accès soient bien implémentés dans les scripts. Ce répertoire est récupéré dans le script d'appel docker(dock.sh) et copié dans la machine de l'utilisateur. Cela lui permet de récupérer chaque fichier créé et de pouvoir les étudier séparément. De plus, en cas de potentielles erreurs, cela permettrait un meilleur diagnostic.

La commande CLONE permet de récupérer le contenu du github à l'identique. En se plaçant dans le bon répertoire, il suffira alors de lancer les scripts désirés. *image de la fonction qui prend le github*

Après s'être placé dans le répertoire " /script" du docker, les scripts se lancent dans l'ordre spécifié par le dockerfile. Ces manipulations sont possibles car dans chaque script, les chemins d'accès sont spécifiques.

Les outputs produits sont placés dans le répertoire "/output" qui sera récupéré sur la machine de l'utilisateur grâce au script dock.sh. Avant l'appel d'eKLIPse et de GATK, python doit être passé en python2 et repassé en python3 après pour les autres scripts.

4.6 Base de données

Notre client n'ayant pas d'informations sur la base de données utilisée au CHU, notre base de données a été réalisée avec PostgreSQL dans un premier temps. Bien que celle-ci soit fonctionnelle, nous avons appris 2 semaines avant la fin du projet qu'elle ne serait pas compatible avec ce qui est utilisé au CHU. Nous avons donc pris la décision de tout recommencer en ayant conscience qu'il y a peu de chances que la base de données soit totalement fonctionnelle au moment du rendu.

Le CHU utilise une base de données en MySQL[19]. L'outil MariaDB[20] est donc utilisé pour construire et interroger notre base de données en MySQL afin qu'elle soit compatible avec celle du CHU.

L'implémentation de la base de données se répartit dans quatre fichiers Python : un pour la création, un pour l'insertion des données, un pour soumettre des requêtes sur la base de données et un fichier main pour gérer l'utilisation de la base de données.

4.6.1 Création

Dans le fichier *Creation_BDD.py*, la fonction *creation()* utilise le module *mariadb* afin d'établir, dans un premier temps, une connexion avec le serveur de base de données. Dans un second temps, un curseur est initialisé et permet d'exécuter les requêtes SQL. Ensuite, est créée, dans cet ordre, la base de données *Variants_mitochondriaux*, la table *Variants*, la table *Annotation* et la table *Echantillons*.

Lors de la création, il faut faire attention à choisir un domaine adéquat pour chaque attribut de chaque table. Il est ainsi important d'implémenter correctement les clés primaires et étrangères.

Si l'une des étapes précédentes échoue, le choix a été fait d'afficher un message d'erreur. Enfin, il y a fermeture du curseur et de la connexion à MariaDB.

4.6.2 Insertion

Le fichier *Insertion_BDD.py* consiste en une fonction *insertion()* qui est en fait un menu faisant appel à de nombreuses autres fonctions. En effet, le code demande dans un premier temps à l'utilisateur dans quelle table s'il souhaite insérer des données. En fonction de son choix, plusieurs manières d'insérer les données seront possibles ou non.

Si l'utilisateur choisit d'insérer des données dans la table *Variants*, il pourra choisir de les entrer soit via un fichier Excel pour les anciens variants soit manuellement pour les nouveaux.

S'il choisit une insertion à partir d'un Excel, l'utilisateur renseigne le path du fichier en input. Ensuite, la fonction *insert_Excel_variants(path_excel)* charge les données présentes dans le fichier Excel dans une dataframe pandas[21]. Puis, cette fonction fait appel à la fonction *remise_en_forme_variants(df)* qui va remettre en bonne forme la dataframe pour que les données puissent être insérées dans la table *Variants*.

Pour remplir les colonnes Pos, Ref et Alt, il faut décomposer la nomenclature des variants. Les variants sont répartis selon 3 cas : substitution, insertion ou délétion. Si la chaîne de caractère du variant commence par le caractère "-", il s'agit d'une insertion. Si la chaîne de caractère du variant se termine par le caractère "-", c'est une délétion. Une découpe de la nomenclature du variant est faite en position, base de référence et base altérée, en parcourant la chaîne de caractère du variant et en vérifiant si les caractères sont des chiffres ou autre chose grâce à la méthode *isdigit()* de Python. Ensuite, il faut vérifier si les variants insérés ne sont pas déjà présents dans la table *Variants* afin d'éviter les redondances et les incohérences.

S'il choisit une insertion manuelle, l'utilisateur renseigne le variant en nomenclature en input. Le code complète ensuite les autres colonnes grâce à l'input renseigné, en décomposant le variant en Pos, Ref et Alt comme précédemment.

Si l'utilisateur choisit d'insérer des données dans la table *Echantillons*, il pourra choisir de les entrer soit via un fichier Excel pour les anciens échantillons soit manuellement pour les nouveaux.

S'il choisit une insertion à partir d'un Excel, l'utilisateur renseigne le path du fichier en input. Ensuite, la fonction *insert_Excel_echantillons(path_excel)* ainsi que la fonction *remise_en_forme_echantillons(df)* vont charger les données présentes dans le fichier Excel dans une dataframe pandas et remettre en bonne forme la dataframe pour que les données puissent être insérées dans la table *Echantillons*.

La difficulté pour la table *Echantillons* est que l'attribut Id doit être unique. Pour cela, le choix a été fait que les échantillons insérés ne seront jamais supprimés. Ainsi, chaque nouvel échantillon pourra avoir un identifiant unique qui sera égal à la longueur de la table plus 1. Il nous faut donc retrouver la longueur de la table *Echantillons* à chaque insertion. Cela est fait grâce à une requête SQL : *SELECT COUNT(*) FROM Echantillons;*

S'il choisit une insertion manuelle, l'utilisateur renseigne le numéro d'échantillon et le variant en nomenclature en input. La colonne Id sera complétée de la même manière que précédemment.

Si l'utilisateur choisit d'insérer des données dans la table *Annotation*, il pourra les entrer via le fichier CSV d'annotations obtenu à l'étape précédente. L'utilisateur doit pour ce faire

renseigner le path du fichier en input. Ne seront alors chargé que les colonnes pertinentes du fichier CSV dans un dataframe pandas. Cela selon l'usage qui sera fait de la base de donnée.

L'insertion des données à partir du fichier d'annotations peut rencontrer quelques problèmes. En effet, il est possible que certaines lignes du CSV contiennent une énumération de variants due à une incertitude lors de l'annotation (voir ligne 9 de la figure 3 ci-dessus). Ces lignes ne pourront être insérées dans la base de données car les valeurs des colonnes ne seront pas compatibles avec le domaine des attributs de la table *Annotation*. On met donc en place une gestion d'erreur qui vérifie que l'on n'a pas d'énumération de variants dans la colonne Alt notamment.

4.6.3 Requêtes

Le biologiste chargé d'interpréter les résultats du pipeline n'aura pas nécessairement des connaissances en SQL suffisantes pour soumettre lui-même des requêtes sur la base de données. Il est donc indispensable de proposer des requêtes pré-enregistrées permettant de répondre aux questions récurrentes que pourra se poser le biologiste ou le médecin en charge de l'interprétation.

Le fichier *Requetes_BDD.py* se compose donc d'un menu qui propose d'effectuer l'une des recherches suivantes, avec un certain input renseigné par l'utilisateur :

- tous les échantillons qui ont une position donnée mutée (input = position)
- tous les variants d'un échantillon (input = numéro d'échantillon)
- tous les variants associés à une maladie (input = maladie)
- tous les variants liés à un gène (input = gène)
- le nombre d'occurrences d'un variant (input = variant)

Une fois que l'utilisateur a choisi la recherche qu'il souhaite effectuer et qu'il a renseigné l'input associé, un appel à la fonction correspondante est fait. Cette fonction lancera alors une requête SQL formatée avec l'input renseigné. Le résultat de la requête est finalement affiché dans le terminal et enregistré dans un fichier Excel.

Par le futur, ce fichier de requêtes pourra être adapté et complété avec d'autres requêtes dont le résultat serait intéressant et utile pour l'interprétation biologique.

4.6.4 Utilisation

Dans le fichier *Utilisation_BDD.py*, il y a importation des fonctions contenues dans les fichiers *Creation_BDD.py*, *Insertion_BDD.py* et *Requetes_BDD.py*.

La fonction *menu()* va alors être exécutée. Elle fait tout d'abord appel à la fonction *exist_BDD()* qui vérifie si la base de données est déjà créée ou non. Si ce n'est pas le cas, la fonction *creation()* créera la base de données.

Une fois sûr de l'existence de la base, un menu propose d'insérer des données (input = 1), de soumettre une requête SQL (input = 2) ou de quitter le programme (input = 0). En fonction du choix de l'utilisateur renseigné en input, le script fera appel soit à la fonction *insertion()*, soit à la fonction *requetes()*, importées depuis les fichiers précédemment cités.

Le menu est réaffiché après l'exécution de chaque fonction tant que l'utilisateur n'a pas décidé de quitter le programme en tapant "0".

5 Conclusion

Plusieurs aspects ont été traités dans ce projet. Le projet final consiste en la création d'un pipeline complet pour l'analyse des maladies mitochondriales en partie contenu dans un conteneur Docker et utilisable sur n'importe quelle machine.

Le pipeline est composé de plusieurs unités connectées entre elles : alignement des reads mitochondriaux sur un génome de référence, appel de variants, annotation de ces variants et obtention d'informations complémentaires grâce aux outils Haplogrep et eKLIPse. La création d'un pipeline permet l'automatisation des analyses, facilitant le travail du biologiste. Au plus il sera automatisé, par exemple avec l'ajout de fonction et la gestion d'erreur, au plus il sera reproductible. L'utilisation d'une docker permet une meilleure portabilité, ainsi il pourra être lancé depuis n'importe quelle machine. Avec ce nouvel outil, l'utilisateur pourra travailler efficacement et faire un diagnostic rapide des maladies mitochondriales pour des patients en attente de réponses. Le projet a également fourni la possibilité d'insérer les diverses informations dans une base de donnée interrogeable.

Après avoir pris du recul sur le pipeline, il est évident qu'il est loin d'atteindre la perfection en terme de rigueur ou même de prise en main. En effet, on peut d'abord citer l'absence de contrôle qualité des reads en amont. Cela sous-entend que l'utilisateur a procédé à ce contrôle au préalable.

De plus, l'utilisation du docker n'est pas optimale. Il aurait été plus juste de faire appel à un outil de gestion de pipeline. Il aurait été intéressant d'utiliser Docker en multi-staging. Le multi-staging permet d'ajouter des étapes de compilation, ainsi chaque étape peut être produite à partir d'une image différente. Cela aurait facilité l'installation de certains logiciels comme VEP. Ainsi l'image prendrait moins de temps à être construite et le temps de téléchargement serait moins long. L'image finale serait moins lourde et cela améliorerait la portabilité.

Enfin, pour aborder les difficultés rencontrés, il est indispensable d'évoquer le fait que l'accès à des serveurs n'a pas été possible. Certains logiciels comme VEP sont particulièrement imposant et très volumineux, ce qui a rapidement mené à la saturation des ordinateurs personnels.

Le projet nous a donné l'opportunité d'approfondir nos connaissances en nous apportant une meilleure compréhension des maladies mitochondriales ainsi qu'une meilleure rigueur dans divers langages informatiques. De plus, nous avons pu acquérir les compétences nécessaires à la conception d'un pipeline et son optimisation. Cela en passant par docker et par l'apprentissage de son utilisation. A l'avenir nos interactions lors de travaux d'équipe seront également améliorées. L'équipe peut ainsi affirmer qu'elle tire de nombreux bénéfices de ce projet et qu'elle en ressort avec de nouveaux atouts.

Veuillez trouver ci-joint le lien du GitHub regroupant tous les scripts implémentés pour ce projet : <https://github.com/yguiberteau/Pipeline-pour-l-analyse-des-maladies-mito>

Références

- [1] R. E. Giles, H. Blanc, H. M. Cann, and C. Wallace aD. Maternal inheritance of human mitochondrial dna. *Proceedings of the National Academy of Sciences*, 77 :6715–6719, 11 1980. ISSN 00278424. doi : 10.1073/PNAS.77.11.6715. URL <https://www.pnas.org/doi/abs/10.1073/pnas.77.11.6715>.
- [2] James E. Davison and Shamima Rahman. Recognition, investigation and management of mitochondrial disease. *Archives of Disease in Childhood*, 102 :1082–1090, 11 2017. ISSN 14682044. doi : 10.1136/archdischild-2016-311370.
- [3] S. Rahman. Mitochondrial disease in children. *Journal of internal medicine*, 287 :609–633, 6 2020. ISSN 1365-2796. doi : 10.1111/JOIM.13054. URL <https://pubmed.ncbi.nlm.nih.gov/32176382/>.
- [4] Yi Shiao Ng, Laurence A. Bindoff, Gráinne S. Gorman, Thomas Klopstock, Cornelia Kornblum, Michelangelo Mancuso, Robert McFarland, Carolyn M. Sue, Anu Suomalainen, Robert W. Taylor, David R. Thorburn, and Doug M. Turnbull. Mitochondrial disease in adults : recent advances and future promise. *The Lancet. Neurology*, 20 :573–584, 7 2021. ISSN 1474-4465. doi : 10.1016/S1474-4422(21)00098-3. URL <https://pubmed.ncbi.nlm.nih.gov/34146515/>.
- [5] Oliver M. Russell, Gráinne S. Gorman, Robert N. Lightowlers, and Doug M. Turnbull. Mitochondrial diseases : Hope for the future. *Cell*, 181 :168–188, 4 2020. ISSN 10974172. doi : 10.1016/j.cell.2020.02.051.
- [6] Serena Dotolo, Riziero Esposito Abate, Cristin Roma, Davide Guido, Alessia Preziosi, Beatrice Tropea, Fernando Palluzzi, Luciano Giacò, and Nicola Normanno. Bioinformatics : From ngs data to biological complexity in variant detection and oncological clinical practice. *Biomedicines*, 10, 9 2022. ISSN 2227-9059. doi : 10.3390/BIOMEDICINES10092074. URL <https://pubmed.ncbi.nlm.nih.gov/36140175/>.
- [7] Markduplicates – gatk. URL https://gatk.broadinstitute.org/hc/en-us/articles/360037052812-MarkDuplicates-Picard-?fbclid=IwAR14KzAxw_eXih-ZCVMVREnztCgEYS6JHj0DRML-7ZiESPda9dBJ-0jLY6U.
- [8] John Doe and Jane Smith. Documentation sur mitomap. *Journal of Mitochondrial Research*, 10 :50–75, 2023.
- [9] William McLaren, Laurent Gil, Sarah E. Hunt, Harpreet Singh Riat, Graham R.S. Ritchie, Anja Thormann, Paul Flicek, and Fiona Cunningham. The ensembl variant effect predictor. *Genome Biology*, 17(1) :122, 2016. doi : 10.1186/s13059-016-0974-4.
- [10] John D. McCarthy and et al. Le choix des transcriptions et des logiciels a un effet important sur l’annotation des variantes. *Nom du Journal*, 2014.
- [11] Haplogrep 3. URL <https://haplogrep.readthedocs.io/en/latest/>.

- [12] David Goudenège, Celine Bris, Virginie Hoffmann, Valerie Desquirit-Dumas, Claude Jardel, Benoit Rucheton, Sylvie Bannwarth, Veronique Paquis-Flucklinger, Anne Sophie Lebre, Estelle Colin, Patrizia Amati-Bonneau, Dominique Bonneau, Pascal Reynier, Guy Lenaers, and Vincent Procaccio. eklipse : a sensitive tool for the detection and quantification of mitochondrial dna deletions from next-generation sequencing data. *Genetics in Medicine*, 21 :1407–1416, 6 2019. ISSN 15300366. doi : 10.1038/s41436-018-0350-8. URL <http://www.gimjournal.org/article/S1098360021016622/fulltext><http://www.gimjournal.org/article/S1098360021016622/abstract>[https://www.gimjournal.org/article/S1098-3600\(21\)01662-2/abstract](https://www.gimjournal.org/article/S1098-3600(21)01662-2/abstract).
- [13] Documentation - bash. URL <https://datascientest.com/bash-tout-savoir/>.
- [14] Python Software Foundation. Python Documentation. <https://www.python.org/doc/>.
- [15] Docker. Docker documentation : Dockerfile reference. <https://docs.docker.com/engine/reference/builder/>. Accessed : May 25, 2023.
- [16] Github docs. URL <https://docs.github.com/fr>.
- [17] Slack Technologies. Qu’est-ce que slack. <https://slack.com/intl/fr-fr/help/articles/115004071768-Qu%E2%80%99est-ce-que-Slack>. Consulté le 25/05/2023.
- [18] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. 3 2013. URL <https://arxiv.org/abs/1303.3997v2>.
- [19] Mysql documentation. URL <https://dev.mysql.com/doc/>.
- [20] Documentation - mariadb.org. URL <https://mariadb.org/documentation/>.
- [21] User guide — pandas 2.0.1 documentation. URL https://pandas.pydata.org/docs/user_guide/index.html.