# Développement d'un pipeline modulaire pour un cluster SLURM partagé adapté à l'analyse du polymorphisme des génomes : Application au tGBS chez les végétaux

Benjamin Loire

# *Rapport de stage de Master 1*

*Développement d'un pipeline modulaire*
*pour un cluster SLURM partagé*
*adapté à l'analyse du polymorphisme des génomes :*
*Application au tGBS chez les végétaux*

**Benjamin Loire**

11/04/2022 - 31/08/2022

INRAE - Institut National de Recherche pour l'Agriculture, l'Alimentation et l'Environnement
Laboratoire INRAE EPGV US 1279 - Étude du Polymorphisme des Génomes Végétaux
2 rue Gaston Crémieux 91057 Évry

Université de Rennes 1 - Master 1 Bioinformatique en Génomique

**Tuteur de stage**

Damien Hinsinger

**Enseignante référent :**

Annabelle Monnier

UNIVERSITÉ DE RENNES 1

# ENGAGEMENT DE NON PLAGIAT

Je, soussigné (e) Benjamin Loire

Etudiant (e) en Master 1 BioInformatique en Génomique

Déclare être pleinement informé (e) que le plagiat de documents ou d'une partie de documents publiés sous toute forme de support (y compris l'internet), constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour la rédaction de ce document.

**Signature**

INSERM U1242 OSS
Equipe PROSAC

Centre Eugène Marquis
Avenue de la Bataille Flandres Dunkerque
35042 Rennes

**Annabelle MONNIER**
annabelle.monnier@univ-rennes1.fr

TÉL. 33 (0)2 23 23 61 14

# Remerciements

Je tiens tout d'abord à remercier mon maître de stage Damien Hinsinger et mon encadrante Aurélie Canaguier pour m'avoir montré le fonctionnement du pipeline pré-existant et accompagné tout au long de ce projet en ayant la patience de répondre à mes questions. Certains obstacles auraient été beaucoup plus durs à dépasser sans votre expérience.

Merci encore à vous deux ainsi qu'à Patricia Faivre-Rampant, Directrice d'Unité de l'EPGV pour la relecture de ce rapport et vos conseils. Merci également de m'avoir permis de me rendre au mini-congrès du Génoscope le 14 juin et au colloque de la Journée des Plateformes de l'IPS2 le 16 juin à Orsay, ces deux évènements étaient très enrichissants.

Je remercie Sandrine Contenot pour avoir jeté un œil à mon code et Romane Guilbaud pour son aide lorsqu'une expertise en R était nécessaire.

Merci à Aurélie Bérard et Isabelle Le Clainche d'avoir été mes bêtas-testeuses et d'avoir supporté avec le sourire tous les petits problèmes du développement.

Un grand merci à Nicolas Wiart (Génoscope) qui a su m'expliquer avec pédagogie et dans le détail le fonctionnement de SLURM et du cluster ainsi que ses bonnes pratiques.

Je remercie aussi mon oncle et ma tante qui m'ont accueillis chez eux pour ces quatre mois de stage. Malgré la route, c'était beaucoup plus agréable d'être dans un environnement familial.

# Introduction

Les nouvelles technologies et méthodes de séquençage permettent l'obtention de grands volumes de données devant nécessairement être analysées par des moyens informatiques [22,29]. Une solution est la mise en place de pipelines, suites d'outils paramétrés permettant d'exécuter une succession de tâches sur un ou plusieurs jeux de données.

La simplicité d'utilisation des pipelines d'analyses devient un enjeu aussi important que leurs performances, en particulier lorsqu'il est question de leur utilisation par des biologistes sans compétences informatiques avancées [5].

Pour être pertinentes, ces analyses doivent obéir à certains critères, synthétisés dans une démarche FAIR (*Findable, Accessible, Interoperable and Reusable*) dans laquelle l'EPGV (unité INRAE d'accueil de ce stage décrite en Annexe 1) est engagée. Elle consiste en la mise à disposition des résultats, la transparence des calculs (le code utilisé doit être accessible et documenté), la portabilité (l'analyse doit pouvoir fonctionner sur des plateformes différentes), et surtout, la reproductibilité des résultats et leur stockage [21]. Les pipelines d'analyse automatisés, en plus de répondre à ce besoin, permettent un gain de temps considérable par rapport à l'exécution manuelle des analyses.

## Automatisation des analyses

Plusieurs outils de structuration (frameworks) de pipelines ont ainsi été développés afin de faciliter le développement d'analyses successives [9,16]. Les frameworks de configuration comme Pegasus [9] reposent sur la déclaration de chaque étape dans un fichier écrit dans un format de configuration comme le YAML *Ain't Markup Language* [30]. Ils sont en général abordables même sans compétence en programmation, mais souffrent du même coup d'une flexibilité moindre dans l'exécution des tâches. Au contraire, les frameworks de convention définissent les tâches dans des fonctions ou des classes à l'aide d'un langage de programmation ou bien d'un langage domaine-spécifique (DSL) comme le fait Snakemake [16]. Il est généralement possible d'ajouter du code de l'utilisateur, ils sont donc plus versatiles mais moins aisément lisibles [17].

Chacun de ces types de framework peut être explicite ou implicite [17]. Un framework explicite demande d'écrire la succession des étapes dans l'un des fichiers du pipeline, là où un framework implicite va inférer cet enchaînement en fonction des noms de fichier d'entrée et de sortie. Le choix d'un framework ou d'un autre repose en partie sur des préférences personnelles

du programmeur et des utilisateurs visés, puisque la plupart des fonctionnalités de base sont partagées entre tous [17].

Malgré leurs différences, ces frameworks ont des objectifs communs, comme la bonne reproductibilité des résultats, une portabilité du pipeline sur plusieurs plateformes, l'évolutivité pour rester performant même avec un nombre important de tâches à exécuter ainsi qu'une flexibilité pour intégrer les scripts et outils de l'utilisateur.

## Génotypage ciblé par séquençage (tGBS)

La puissance apportée par le séquençage de nouvelle génération (NGS) est considérable et a permis une augmentation du nombre de données génomiques produites et de leur utilisation [22]. Si le génotypage par séquençage complet du génome est la méthode apportant une meilleure vue d'ensemble du polymorphisme : SNP (*Single Nucleotide Polymorphism*) et variations structurales, elle est difficilement accessible à grande échelle en raison de son coût. En revanche, l'approche du génotypage de SNP ciblés par séquençage permet d'effectuer un appel de variants au SNP cible et dans les régions flanquantes [23]. Il s'agit d'une approche qui permet, à coût égal avec le séquençage d'un génome entier, d'obtenir une profondeur compatible avec un génotypage de qualité d'une population dans les régions d'intérêt. Ceci se vérifie en particulier pour les grands génomes de certaines plantes cultivées comme le maïs en ciblant la quantité de séquences nécessaire [27].

Les technologies NEBNext *Direct Genotyping Solution* (ci-après abrégé en "NEBNext") [2] et Allegro *Targeted Genotyping* V2 (ci-après abrégé en "Allegro") [27] sont deux méthodes pour effectuer un séquençage de SNP ciblés, toutes deux réalisant l'enrichissement en SNP cibles sans PCR (Tableau 1). La technologie NEBNext est basée sur un enrichissement par capture des régions cibles à l'aide de sondes biotinylées [2], là où Allegro emploie l'approche SPET (*Single Primer Extension Technology*) : l'enrichissement en régions cibles se fait par l'extension d'une sonde unique située en 5' proximal du SNP cible [27]. Dans les deux technologies, le fournisseur recommande l'usage de deux sondes par SNP, pour une robustesse accrue du génotypage.

Ces technologies requièrent une analyse bioinformatique complexe afin d'inférer les génotypes. Ce besoin informatique pourrait limiter la diffusion de ces technologies auprès d'utilisateurs non-bio-informaticiens [5]. Il s'agit donc dans ce stage de M1 d'automatiser et de rendre ergonomique un pipeline d'alignement NGS et de génotypage de SNP cibles développé par l'EPGV pour traiter les données de séquençage issues des deux technologies précitées.

Tableau 1 : Spécifications et principes généraux des technologies NEBNext et Allegro, (d'après le projet INRAE-IB21, *GBS expert*)

| Technologie | Allegro targeted Genotyping *(NuGen/Tecan)* | NEB Next Direct Genotyping Solution (*NEB)* |
|---|---|---|
| **Réduction génomique / enrichissement basé(e) sur** | SPET (*Single Primer Extension Technologie*) : extension d'amorce | Capture |
| **Nombre de sondes par SNP ciblé** | 2 sondes indépendantes/Marqueurs | 2 sondes indépendantes /Marqueurs |
| **Zone du design des sondes** | Région proche du SNP | Région proche du SNP |
| **Type de séquençage** | Illlumina<br>SE 100/150 ou PE 2*150 | Illumina<br>SE 100/135 |
| **Nombre de cibles** | Jusqu'à 50 000 (voire plus) | Jusqu'à 5 000 |
| **Multiplexage d'individus** | jusqu'à 3 072 | jusqu'à 9 216 |
| **Balance des sondes** | non | oui |
| **Tag moléculaire** | oui (optionnel) | oui |
| **Analyse des données** | Pipeline d'outils publics à implémenter | Pipeline d'outils publics à implémenter |
| **Déploiement en AgriGénomique** | +++ | + |
| **Publications en AgriGénomique** | ++ | + |

Notre pipeline automatisé n'utilise pas de framework existant mais en reprend de nombreuses fonctionnalités. Il ne s'agit pas non plus d'un nouveau framework, mais d'un cadre modulaire se positionnant entre le pipeline de configuration explicite et celui de convention, pensé pour un cluster utilisant SLURM. Le développement répondait à ces trois contraintes :

- Développement d'un pipeline généraliste pouvant s'adapter à d'autres méthodes et technologies
- Génération de métriques complètes et de log permettant de suivre l'avancement de chaque projet
- Facilité d'accès et d'utilisation pour des non-bioinformaticiens

STAMPS (Simple Targeted GBS Automated and Modular Pipeline for SLURM) est un pipeline développé en langage python, qui se positionne entre le framework de convention explicite et le framework de configuration.

# Matériel et méthodes

## Environnement de travail

Le cluster de calcul utilisé lors du développement et des analyses du pipeline est celui du CEA (Génoscope et CNRGH). Appelé inti, il tourne sous Linux CentOS 7.9.2009 compatible RedHat et traite les soumissions avec le logiciel de gestion de ressources SLURM 20.11.9.

SLURM est un gestionnaire de ressources et ordonnanceur libre, utilisé dans de nombreux supercalculateurs à travers le monde [15,25]. L'outil Environnement Modules [6] a été utilisé pour charger les différents outils dans les environnements, de façon à laisser la gestion et le versionnage des outils aux administrateurs du cluster.

Ce cluster utilise un système de partitions possédant des contraintes administratives et techniques. La partition par défaut, "normal", n'autorise pas de tâche de plus de 24 h et possède entre 7,4 et 15,2 Gio de mémoire vive par cœur. Elle est dotée de 47 nœuds et 1 236 cœurs, offrant une capacité de calcul conséquente.

## Jeux de données de test

Afin de confirmer la versatilité du pipeline dans les différentes conditions auxquelles il sera confronté, trois jeux de tests différents ont été utilisés (Tableau 2). En plus des éléments spécifiques à chaque espèce, cette diversité a permis de couvrir les paramètres suivants : réalisation d'un séquençage single-end ou paired-end, à partir de librairies construites en utilisant la technologie Allegro ou NEBNext, ainsi qu'un nombre d'échantillons et de cibles variables.

Tableau 2 : Caractéristiques ayant un impact sur le fonctionnement du pipeline pour les différents jeux de données de test.

| Projet | INVITE [13] | RésiLens [24] | Amaizing [1] |
|---|---|---|---|
| **Espèce** | Tomate (*Solanum lycopersicum* L.) | Lentille (*Lens culinaris* Medik.) | Maïs (*Zea mays* L.) |
| **Technologie** | Allegro | Allegro | NEBNext |
| **Type de lecture** | Single-End | Paired-End | Single-End |
| **Nombre d'échantillons** | 384 | 336 | 768 |
| **Nombre de SNP** | 19 998 | 23 208 | 106 |
| **Nombre de canaux de séquençage** | 2 | 2 | 1 |
| **Taille du génome / du plus grand chromosome** | 950 Mb / 91 Mb | 4 Gb / 614 Mb | 2,4 Gb / 307 Mb |

## Présentation du pipeline initial

Le pipeline à automatiser reposait déjà sur de nombreux outils de manipulation de séquences, de génotypages et de caractérisations des SNP. Les outils et le langage suivants ont été utilisés :

- **SAMtools** 1.15.1 [19] (Conversion des fichiers, indexage et génération des métriques)
- **Picard** 2.26.9 [4] (Préparation des fichiers et génération des métriques).
- **BWA mem** 0.7.15 [18] (Alignement)
- **Probefilter** [8] (Marquage des séquences des sondes avec la technologie Allegro)
- **GATK** 4.2.3.0 [20] (Appel de variant et création des bases de données génomiques)
- **R** 4.1.1 (Assemblage des métriques et génération des graphiques)

### Données d'entrée

Le pipeline accepte en entrée des fichiers FASTQ générés par le séquençage de librairies NEBNext ou Allegro, préférentiellement nettoyés ; une séquence de référence au format FASTA; la position des SNP cibles au format BED et une ou plusieurs listes d'échantillons devant comporter des colonnes prédéfinies. Ces fichiers sont ensuite validés avant de générer des index du fichier de référence et le fichier des régions cibles.

### Alignement

L'alignement des reads sur la séquence de référence est effectuée par BWA-MEM avec l'option -p en cas de reads paired-end. La suite d'outils Picard est ensuite utilisée pour (i) marquer les reads dupliqués avec MarkDuplicates et produire un fichier de métriques; (ii) appliquer des read groups sur chaque fichier BAM avec AddOrReplaceReadGroup; (iii) fusionner les alignements de chaque canal avec MergeSamFiles uniquement si besoins; (iv) générer des métriques d'alignements avec CollectHsMetrics. D'autres fichiers de métriques sont produits à l'aide des outils SAMtools bedcov et flagstats.

Pour la technologie Allegro, Probefilter est utilisé pour passer en bases flottantes les 40 premières bases du read 1 (séquence de la sonde) qui seront ignorées lors de la détection de variants. L'option -p est utilisée en cas de paired-end, permettant le même processus pour les bases du read 2 chevauchant la séquence de la sonde (cas des petits inserts) [8].

### Appel de variants

GATK HaplotypeCaller est utilisé pour appeler les variants sur la totalité de la référence avec les options suivantes : --min-pruning 3 --max-num-haplotypes-in-population 200 --emit-ref-confidence GVCF --max-alternate-alleles 1 --contamination-fraction-to-filter 0.0 --native-

pair-hmm-use-double-precision true --max-reads-per-alignment-start 0. Il s'agit d'une longue étape (production des fichiers GVCF individuels), qui représente la majeure partie de la durée du pipeline.

## Appel de génotypes

Le génotypage se fait en deux temps : tout d'abord GATK GenomicsDBImport crée des bases de données génomiques à partir des fichiers GVCF de chaque échantillon. Puis un fichier GVCF est créé pour chaque chromosome avec GATK SelectVariants, regroupant la totalité des échantillons. Ces fichiers sont ensuite réunis en un GVCF unique par Picard GatherVcfs et indexés par GATK IndexFeatureFile. À partir de ce fichier combiné, trois fichiers VCF contenant les génotypes sont extraits par GATK GenotypeGVCFs : un fichier contenant tous les variants, un autre contenant toutes les positions cibles, qu'elles soient monomorphes ou polymorphes, et un dernier contenant uniquement les SNP *de novo*. Il s'agit de tous les SNP détectés en dehors des positions cibles. Ces dernières sont toutes conservées grâce à l'option --force-output-intervals $TARGETS et éliminées avec --exclude-intervals $TARGETS, où $TARGETS est le fichier BED contenant les positions des SNP cibles.

L'appel de variant n'autorise qu'un seul allèle alternatif par position et par échantillon, mais la réunion de centaines d'individus peut aboutir à une position ayant plusieurs allèles alternatifs, ce qui pose problème pour certains outils. Les SNP qui ne sont pas bialléliques sont ignorés pour les étapes suivantes. Les génotypes restants sont filtrés par défaut via leur profondeur (Depth < 10x) ; la qualité de leur génotype (Genotype Quality < 20) ; et leur fréquence allélique (0.2 < Allele Balance < 0.8). Les génotypes ne passant pas les deux premiers filtres sont mis en données manquantes et les fréquences alléliques permettent l'assignation d'un génotype homozygote référence, homozygote alternatif ou hétérozygote. Dans le cas d'une espèce diploïde homozygote (cas des lignées de plantes cultivées, *e.g.* chez la tomate ou le maïs), un individu est dit homozygote alternatif si la fréquence de l'allèle alternatif est supérieure ou égale à 0.8, et homozygote référence si elle est inférieure à 0.2. Dans le cas d'une espèce diploïde hétérozygote, ces seuils seront adaptés.

Enfin, des matrices de génotypage sont obtenues avec VCFtools. Une valeur de 0/0 indique un homozygote référence, 0/1 un hétérozygote, 1/1 un homozygote alternatif et ./. une donnée manquante par absence d'alignement ou après application des filtres définis.

## Génération des métriques

Différentes métriques qui permettent de suivre la qualité des données produites sont assemblées sous forme de tableaux récapitulatifs ou de graphiques avec R e.g. les informations

sur les données manquantes par SNP et échantillons, couverture sans duplicats et graphique des distances entre SNP *de novo* et SNP cibles.

## Méthodes utilisées pour l'automatisation

Pour réaliser l'automatisation, le choix s'est porté sur du python 3.8.11 standard. Ce choix a été motivé par la disposition du langage à fonctionner aussi bien sur Windows que sur Linux, son efficacité à gérer des tâches simples, sa facilité d'apprentissage ainsi que la popularité du langage - pour un maintien dans le temps. L'automatisation repose sur le programme python d'une part et un fichier de configuration TOML 1.0.0 [28] spécifique à chaque projet. Le TOML a été favorisé par rapport à d'autres formats de configuration plus communs comme le JSON ou le YAML car il est plus facilement lisible et moins lourd. Les modules python suivants ont été utilisés : tomli 2.0.1 pour la lecture du fichier de configuration et tomli_W pour l'écriture du fichier de configuration à archiver ; pandas 1.4.2 pour manipuler les tableaux et matrices ; pytest 7.1.2 pour réaliser des tests unitaires ; et rmdawn 0.1.2 pour la génération dynamique du rapport. Le contrôle de version du code source a été réalisé *via* le logiciel git 1.8.3.1 afin de suivre le développement et de pouvoir publier le code à tout moment sur un dépôt distant (https://forgemia.inra.fr/epgv/stamps/).

Les tests unitaires évaluent les fonctions atomiques d'un programme à l'aide de données contrôlées, afin d'en éprouver la robustesse. Ils ont été écrits dès que nécessaire et sont exécutés lors de chaque lancement du pipeline dans le but de détecter toute erreur de programmation ou d'incompatibilité avec les librairies externes. Ensuite, le fichier de configuration est validé, en évaluant l'existence des chemins et le format des fichiers attendus. Enfin, les modules nécessaires au pipeline sont chargés dans l'environnement, puis l'utilisateur est invité à exécuter le pipeline.

La parallélisation d'un nombre important de travaux a été effectuée à l'aide du framework Pegasus pour sa capacité de parallélisation et d'attribution des ressources efficaces sur plusieurs nœuds [10]. Ainsi, les tâches conséquentes sont lancées dans une tâche générale Pegasus qui prend ensuite le relais de SLURM pour répartir les sous-tâches de façon efficace dans l'espace qui lui est alloué.

# Résultats

## Développement de STAMPS

Les étapes successives sont explicitement indiquées dans un fichier de configuration et la flexibilité d'un langage de programmation est pleinement exploitée. Le fichier de configuration contient beaucoup de valeurs par défaut pour permettre un lancement rapide, mais il est entièrement personnalisable pour se conformer aux besoins et préférences de l'utilisateur.

Il est composé de trois fichiers principaux : le fichier de configuration au format TOML contient les informations du projet et des paramètres fixés pour chaque étape à réaliser ; le script python pipeline_steps.py possède les fonctions de chaque étape et réalise les tâches légères ou crée la commande soumise au cluster ; enfin, le script pipeline.py (Figure 1) fait le lien entre les deux fichiers précédents et le cluster : il appelle les fonctions de pipeline_steps.py en fonction des paramètres du fichier de configuration et récupère les commandes pour les soumettre au cluster via des appels sbatch (Figure 2).

```
for step in config["Pipeline"]:
  current_step = config["Pipeline"][step]
  if isinstance(current_step,dict) and current_step["Run"]:
   samplelist = utils.Get_Samples_List(config["Pipeline"][step]) else []
   if utils.Check_Final_Outputs(config, step, samplelist=samplelist):
     log_message = f"Step {step} looks done, skipping..."
     continue
   command = getattr(steps, step)(config) #Call associated function
   dependencies_ID = []
   for dep in current_step["Dependencies"]:
     if dep in Job_Tracker: dependencies_ID.append(Job_Tracker[dep])
     params = config["Slurm_Parameters"][step] if step in config["Slurm_Parameters"]
else None
     command = utils.Format_Command(command, params, dependencies_ID)
     if "sbatch" in command:
      if config["Pipeline"]["Pegasus_Script"] in command:
        common_script = os.path.join(config["Pipeline_Dir"],"scripts",
f'common_{config["Project"]["Name"]}.sh')
        command = command + f' -v "{config["Pipeline"]["Pegasus_Rescuer_Script"]} -c
{config["Pipeline"]["Config_Output_File"]} -s {step} -o
{os.path.join(dirout,f"Pegasus_step_{stepname}.err")} -r
{config["Pipeline"][step]["Pegasus_Reboot_Limit"]} -d {common_script}"'
        result = subprocess.run(f"{command}", shell=True, capture_output=True,
text=True) #Send sbatch command to the cluster
        Job_Tracker[step] = ":".join(jobids)
```

Figure 1 : Extrait du script pipeline.py. Pour chaque étape indiquée dans le fichier de configuration, ce code appelle la fonction associée avant d'envoyer la commande sbatch adapté au cluster.

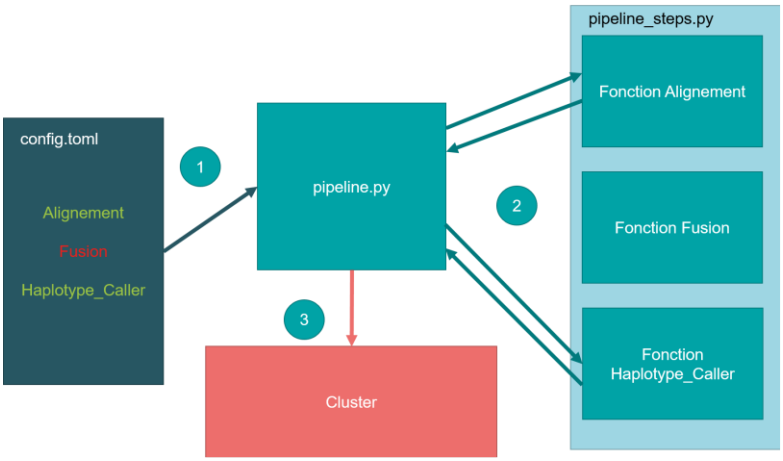


Figure 2 : Schéma du fonctionnement technique de STAMPS.

En plus de ces fichiers principaux, il en existe trois autres : defaults.toml, qui permet d'attribuer des valeurs par défaut à certains champs du fichier de configuration, configuration.py qui traduit le fichier de configuration au format TOML en un dictionnaire interprété par python et valide les informations contenues, et utils.py qui contient des fonctions utiles partagées entre plusieurs scripts.

Les travaux sont tous soumis en même temps et chaînés par dépendance via l'option SLURM *-d afterok*. Il revient ensuite aux scripts BASH de chaque job de vérifier que les fichiers attendus sont bien présents.

Afin de faciliter l'écriture du fichier de configuration, un système de variables a été créé. Cela évite de réécrire plusieurs fois un long chemin de fichier qui a déjà été référencé (notamment dans les valeurs des clefs principales Pipeline.Main_Project_Dir ou Pipeline.Samples_Dir), ou d'adapter des paramètres à des valeurs changeant régulièrement comme la date. Au lancement du pipeline (par la commande ~/STAMPS/Launcher.sh -c config.toml depuis le répertoire projet), ces variables sont remplacées par les valeurs qu'elles référencent, et un fichier "traduit" est généré afin de figer cette configuration dans le temps.

Le pipeline s'attend à une structure identique pour chaque projet et est capable de la créer au lancement via le script init.py. Il est possible d'ignorer cette structure en écrivant des chemins personnalisés, mais ce n'est pas recommandé. À chaque étape, des fichiers de suivis sont produits, ainsi que des métriques complètes des résultats des tâches effectuées.

Un certain nombre de fonctionnalités des frameworks de pipelines ont été implémentées (Tableau 3). STAMPS est ainsi capable de déterminer si une étape est achevée en vérifiant la présence de fichiers finaux pour cette étape. Cela permet d'éviter de relancer des travaux déjà terminés et facilite la reprise d'erreurs. Ce système fonctionne pour chaque échantillon pour

les étapes comprenant de nombreux travaux parallèles (*e.g.* travaux lancés *via* Pegasus). Ce fonctionnement permet également de préserver de l'espace disque, en supprimant les fichiers intermédiaires lorsqu'ils ne sont plus utiles. Lorsque tous les fichiers d'une étape sont intermédiaires, la création d'un fichier vide servant d'indicateur de terminaison permet de conserver ce principe.

Tableau 3. Comparatif de différents frameworks de pipeline existants, montrant les caractéristiques pertinentes pour notre implémentation. Les caractéristiques de STAMPS sont incluses en dernière colonne.

| Framework | Ruffus | Nextflow | Snakemake | Pegasus | STAMPS |
|---|---|---|---|---|---|
| **Langage** | Python | DSL/Groovy | DSL/python | Python | Python |
| **Mécanisme** | Convention explicite | Convention implicite | Convention implicite | Configuration explicite | Convention explicite |
| **OS** | Tous | Linux/MacOS | Linux/MacOS | Tous | Linux/MacOS |
| **Compatible SLURM** | + | ++ | ++ | ++ | ++ |
| **Compatible cluster inti** | - | + | + | ++ | +++ |
| **Reprise d'erreur** | - | ++ | - | ++ | ++ |
| **Parallélisation** | ++ | ++ | ++ | ++++ | ++++ |
| **Mise à l'échelle** | ++ | ++ | ++ | ++ | ++ |
| **Point de contrôle** | ++ | ++ | ++ | ++ | ++ |
| **Référence** | [12] | [11] | [16] | [10] | Ce projet |

La reproductibilité des analyses est assurée par la génération d'une archive statique, contenant tous les scripts utilisés par le pipeline et un fichier de configuration complet possédant les numéros de version des outils. Il est donc possible de relancer une analyse dans des conditions rigoureusement identiques, même des années plus tard, à condition que les outils et leurs versions soient toujours présents.

## Génération du compte rendu

Une fois les résultats et métriques obtenus, un rapport détaillé est produit automatiquement d'après un modèle à l'aide du module python rmdawn. Le rapport est au format Rmarkdown et généré à partir de fichiers textes, scripts et graphiques indépendants, qui peut ensuite être rendu au format HTML, PDF ou Word. Le rapport lui-même est modulaire, puisqu'il est possible de choisir quelles sections inclure.

Le script report_generation.py s'appuie sur des fichiers textes modèles (type Rmarkdown) contenant de nombreuses variables à la façon d'un texte à trous, et traduit toutes les variables du texte modèle avec des valeurs réelles pour chaque section sélectionnée. Il utilise ensuite rmdawn pour assembler le rapport en Rmarkdown et faire le rendu dans le format souhaité. Rmdawn nécessite de séparer en fichiers indépendants l'en-tête en YAML, le texte, les blocs de codes et les images, ce que le script fait préalablement.

## Utilisation des jeux de données : développement et tests

Les trois jeux de données (Tableau 2) ont eu de divers rôles pour l'automatisation du pipeline, selon leurs spécificités. Le jeu initialement utilisé pour le développement était le sous-set de validation expérimentale de 40 individus du projet INVITE. Il a également servi à évaluer la répétabilité des analyses sur 8 répétitions indépendantes, montrant que les matrices de génotypes générées sont identiques à 100 % entre les répétitions. Une montée en échelle a ensuite été réalisée sur les données complètes des projets INVITE et RésiLens, menant à une amélioration et optimisation du code. De plus, les caractéristiques de la référence du projet RésiLens (taille des chromosomes) ont permis d'identifier des limites de l'indexage BAI et d'adapter le pipeline en conséquence. Enfin, les données du projet Amaizing ont servi à confirmer la capacité de STAMPS à fonctionner avec des caractéristiques différentes des deux autres projets (*e.g.* technologie, séquençage monocanal).

La comparaison du lancement de ces projets avec et sans automatisation de ce pipeline montre un gain de temps important. En effet, là où une exécution manuelle nécessitait environ une semaine d'attention quasi-constante d'un bio-informaticien pour mener à bien un projet (de la réception des données à l'obtention des matrices de génotypage), moins d'une demi-journée (évaluation réalisée lors des sessions "Bac à sable") suffit maintenant à un non-bioinformaticien pour préparer et lancer STAMPS, qui s'exécute ensuite sans besoin d'interaction pendant 1 à 5 jours.

# Discussion

## Choix méthodologique

Lors du choix du type de pipeline, plusieurs problèmes sont apparus. La plupart des frameworks de pipelines populaires (Tableau 3) ne sont pas compatibles tel quel avec un cluster de calcul tel qu'inti. Puisqu'ils lancent des outils pouvant durer très longtemps, ils sont obligés

de rester actifs pour toute la durée du traitement (en occupant au minimum un cœur), parfois plusieurs jours, ce qui les amène à dépasser les limites administratives d'inti.

D'après les contraintes administratives du cluster inti, une tâche sur la partition "normal", celle possédant le plus de cœurs, ne peut excéder 24 h, elle est automatiquement arrêtée passé ce délai. Ces limites imposées par l'administrateur contraignent l'utilisateur à une meilleure gestion de ses ressources, conforme à l'usage des supercalculateurs du CEA, et une optimisation des programmes. En utilisant un outil de pipeline classique, il serait techniquement possible de demander un cœur sur les partitions "xlarge" ou "xxlarge" (dont la limite est de 7 jours, mais ils sont réservés aux tâches consommant plus de 300 Go de mémoire) mais ces partitions sont petites et ce n'est pas leur usage recommandé, ou bien de lancer sur une machine frontale en y consommant des ressources et en prenant le risque de perdre la connexion entre l'outil et les jobs lancés. Ces méthodes sont contraires aux bonnes pratiques du cluster CEA et immobilisent des ressources importantes pour des programmes majoritairement inactifs.

Après avoir considéré les avantages et inconvénients de chaque approche, il a été décidé de développer STAMPS en python pur, sans utiliser de framework. Cette méthode, qui demande un plus long temps de développement, a les avantages de donner un contrôle total sur les opérations effectuées et d'être compatible avec les contraintes administratives du cluster.

Un point important était de permettre l'extension future du pipeline. Compte tenu du temps relativement court pour la mise en place de l'automatisation, une attention particulière a été portée à la modularité. Dans le cadre actuel, il est très simple d'ajouter une nouvelle étape au pipeline. Il suffit d'ajouter la section correspondante dans le fichier de configuration avec les paramètres particuliers en respectant les conventions de nommage des clefs ; d'écrire la fonction dans le script pipeline_steps.py qui formate la commande à soumettre et d'ajouter le script BASH effectuant les opérations.

Le temps pour développer une nouvelle étape n'est pas négligeable : une demi-journée au moins est nécessaire avec une bonne maîtrise du code, si le script BASH n'est pas déjà écrit. Cependant, une fois l'étape ajoutée, il n'y a plus besoin de modifier autre chose que le fichier de configuration. Cette façon de faire permet un contrôle total et transparent sur le déroulement du pipeline. Afin de faciliter de futurs développements, j'ai mis à disposition un certain nombre de fonctions utilitaires, y compris certaines destinées à être empaquetées dans un module python général destiné à être publié.

## Modules

Travailler avec les modules était une nécessité pour ne pas devoir maintenir à jour les nombreux outils utilisés en permanence. Malgré le confort apporté par cette approche pour le maintien du pipeline sur le long terme, cela est venu avec des contraintes de vérification supplémentaires. En effet, les modules peuvent dans de rares cas être disponibles, mais pas propagés dans les volumes attendus, ce qui implique une vérification préalable de l'existence du module et de sa disponibilité. Cette étape permet également de rester vigilant sur les mises à jour des outils, parfois accompagnées de changements d'options et paramétrage.

## Limites du format BAI

Un problème critique est apparu lors des essais avec le jeu de données du projet RésiLens. L'alignement échouait systématiquement à cause de l'impossibilité pour SAMtools d'indexer les fichiers BAM. Il s'est avéré que le format BAI ne permet d'indexer des contigs que jusqu'à 2^29 bases, soit environ 512 Mégabases [26]. Or, le génome de référence de la lentille comporte deux chromosomes dépassant cette limite (Chromosomes 1 et 2, respectivement de 538 et 614 Mégabases). Il a donc été nécessaire d'utiliser un autre format d'indexage, le CSI, capable d'indexer 2^31 bases ou 2,15 Gigabases [26].

Le format d'indexage classique des fichiers VCF compressés, le TBI, présente la même limite. SAMtools permet là aussi d'utiliser le format CSI, mais pas GATK IndexFeatureFile qui infère automatiquement le type d'index supposément adapté au format du fichier. La solution trouvée a été de décompresser les fichiers VCF afin que l'indexage se fasse au format IDX, autorisant de plus longs chromosomes. Ces fichiers pouvant être volumineux sont compressés après l'analyse.

Cette solution n'est cependant pas absolue compte tenu de la limite du format CSI. Il existe de grands génomes comportant peu de chromosomes qui dépasseraient alors le seuil limite. Par exemple, le génome de la fève *Vicia faba* contient 6 chromosomes pour un total de 13 Gigabases [9], il est donc certain que son plus grand chromosome dépasse les 2,15 Gigabases. Une solution envisagée dans un tel cas est de séparer le chromosome en deux parties dans la séquence de référence pour le maintenir sous cette limite, à moins que de nouveaux formats d'indexage plus performants ne voient le jour.

## Stockage

Une problématique conjointe à l'analyse d'un grand nombre de données génomiques est celle de leur stockage [7]. Même en tGBS, les fichiers d'alignement BAM et d'appel de variant

VCF sont d'une taille conséquente, celle-ci augmentant plus ou moins linéairement avec le nombre d'individus et de variants. Par exemple, un projet de la taille d'INVITE occupe 146 Go d'espace disque et RésiLens 307 Go. Bien que ces dimensions restent dans le domaine du raisonnable, réduire au maximum la place qu'occupent ces fichiers est une préoccupation quotidienne, ne serait-ce que pour pouvoir réaliser plusieurs analyses simultanément. Il faut alors dissocier les notions d'espace utilisé par les analyses et d'espace devant être disponible. Cette différence est particulièrement flagrante pour les projets possédant de gros génomes et de longs chromosomes, forçant à travailler avec des fichiers VCF non compressés comme Résilens. Après la création du GVCF combiné, le dossier du projet contenait 806 Go de données, dont 370 Go uniquement pour ce GVCF. Une fois compressé, ce fichier était réduit à 12 Go. En considérant les résultats de RésiLens comme une estimation haute des analyses courantes de l'EPGV, il a été déterminé que pour fonctionner sur des génomes possédant de longs chromosomes, 1 To d'espace disque doit être disponible, et la moitié de ce volume pour le stockage final.

## Ergonomie et simplicité d'utilisation

L'utilisation du pipeline a été énormément simplifiée : il n'est plus nécessaire de modifier chaque script à la main et de les lancer manuellement, il suffit de remplir le fichier de configuration et d'écrire la ligne de commande pour exécuter toute l'analyse. Cependant, ce n'est pas encore complètement ergonomique, et demande un temps d'apprentissage afin de se familiariser avec le remplissage du fichier ainsi qu'une connaissance de la ligne de commande. Une amélioration de l'accessibilité consisterait à le rendre exécutable sans aucune connaissance préalable et avec une formation minimale, ce qui n'est pas encore le cas.

C'est pourquoi une attention particulière a été portée à l'affichage de messages d'erreurs et d'avertissement lisibles et explicites, ainsi qu'à la production de fichiers informatifs rapportant l'avancement du pipeline à plusieurs niveaux de détail : de la vue d'ensemble indiquant quelles étapes ont été effectuées avec le fichier Suivi_pipeline.txt, aux logs techniques précis fournis par les logiciels utilisés, en passant par une vue intermédiaire spécifique à chaque étape, plus détaillée.

Tout au long du développement, un suivi a été effectué avec les membres de l'équipe wetlab qui seront les principaux utilisateurs de l'outil. Plusieurs sessions "bac à sable" ont permis de pointer quelles fonctionnalités étaient les moins intuitives et d'identifier des bugs, par exemple lors de plusieurs utilisations concurrentes du programme. Ce suivi a été l'occasion

de préciser la direction du développement et de garder cet enjeu de simplicité tout au long du stage.

# Conclusion

Le pipeline tGBS automatisé permet à des non-bioinformaticiens de lancer un pipeline complexe facilement, avec un temps d'apprentissage relativement faible. Le cadre du pipeline est pensé pour fonctionner dans un cluster partagé utilisant SLURM, et permet une grande parallélisation des tâches entre les nœuds, contrairement à une grande partie des frameworks usuels. Toutefois, dans un environnement avec un seul nœud ou sans nécessité de partager les ressources avec plusieurs équipes, un framework plus classique sera plus adapté.

Le pipeline est automatisé et fonctionnel, mais de nombreuses améliorations sont encore possibles. Sa modularité permet d'ajouter de nouvelles fonctionnalités et outils avec un coût de développement restreint à la fonctionnalité en question puisque l'intégration est automatique. Ces ajouts peuvent être nécessaires selon les besoins des utilisateurs, car le pipeline en l'état n'accepte en entrée que des fichiers FASTQ qu'il ne nettoie pas.

Ensuite, pour réduire la taille des fichiers d'alignements (occupant 72 Go pour INVITE et 178 Go pour RésiLens), il a été considéré de les convertir en fichier CRAM [14]. Il s'agit d'un format compressant les fichiers alignés en fonction d'une référence, en n'enregistrant que les positions où les reads diffèrent de la référence. La dernière version de ce format, CRAM 3.1, permet des fichiers plus petits de 50 à 70 % par rapport au fichier BAM correspondant [3]. Il s'agit donc d'un gain d'espace disque conséquent, qui permettrait de fait d'augmenter l'espace disponible et les capacités d'analyse de la plateforme pour faire fonctionner plusieurs projets en parallèle.

Par ailleurs, l'utilisation actuelle du pipeline consiste en un fichier de configuration à éditer manuellement et d'un lancement en ligne de commande, qui n'est toujours pas pleinement adapté à un public sans compétence informatique. L'intégration de ces étapes dans une interface graphique, de bureau ou web, augmenterait l'accessibilité du pipeline à de nouveaux utilisateurs. Afin de conserver sa versatilité et sa modularité, le code source devrait toutefois être toujours accessible.

Enfin, une amélioration importante serait d'ajouter la possibilité de prévoir dynamiquement lors de la soumission d'un travail au cluster les besoins du logiciel et d'attribuer les ressources optimales en fonction de l'utilisation actuelle du cluster. Cela représente plusieurs défis à relever, en premier celui de l'anticipation dynamique de la durée

d’un programme complexe. Ensuite, puisque tous les travaux sont soumis simultanément, la situation du cluster pourra avoir évolué au moment où une tâche commencera à s’exécuter, l’idéal étant alors d’adapter les demandes de cette tâche.

# Références:


1. Amaizing. « Contexte et objectifs ». Consulté le 12 août 2022. https://amaizing.fr/projet/contexte-et-objectifs/.
2. Bjoern Textor, Amy B.Emerman, Kruti M. Patel, Sarah K. Bowman, Scott M. Adams, Brandan S. Desmond, Jonathon S. Dunn, et al. « High-Throughput Screening of 2300 Genetic Markers in *S. lycopersicum* using the NEBNext Direct Multiplexed Genotyping Approach ». New England Biolabs, s. d. https://www.neb.com/-/media/posters/document/high-throughput-screening-of-2300-genetic-markers-in-s-lycopersicum-using-the-nebnext-direct-multiplexed-genotyping-approach.pdf?la=en&rev=9b0ea49912ab488a928444b0714a14d1.
3. Bonfield, James K. « CRAM 3.1: advances in the CRAM file format ». *Bioinformatics* 38, nº 6 (2022): 1497- 1503. https://doi.org/10.1093/bioinformatics/btac010.
4. « broadinstitute/picard ». Java. 2014. Reprint, Broad Institute, 3 août 2022. https://github.com/broadinstitute/picard.
5. Carvalho, Benilton S., et Gabriella Rustici. « The challenges of delivering bioinformatics training in the analysis of high-throughput data ». *Briefings in Bioinformatics* 14, nº 5 (2013): 538- 47. https://doi.org/10.1093/bib/bbt018.
6. « cea-hpc/modules ». Tcl. 2017. Reprint, cea-hpc, 10 août 2022. https://github.com/cea-hpc/modules.
7. Cochrane, Guy, Blaise Alako, Clara Amid, Lawrence Bower, Ana Cerdeño-Tárraga, Iain Cleland, Richard Gibson, et al. « Facing growth in the European Nucleotide Archive ». *Nucleic Acids Research* 41, nº D1 (2013): D30- 35. https://doi.org/10.1093/nar/gks1175.
8. Cooper, James W., Michael H. Wilson, Martijn F. L. Derks, Sandra Smit, Karl J. Kunert, Christopher Cullis, et Christine H. Foyer. « Enhancing faba bean (*Vicia faba* L.) genome resources ». *Journal of Experimental Botany* 68, nº 8 (2017): 1941- 53. https://doi.org/10.1093/jxb/erx117.
9. Deelman, Ewa, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, et al. « Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems ». *Scientific Programming* 13 (2005): 219- 37. https://doi.org/10.1155/2005/128026.
10. Di Tommaso, Paolo, Maria Chatzou, Evan W. Floden, Pablo Barja, Emilio Palumbo, et Cedric Notredame. « Nextflow enables reproducible computational workflows ». *Nature Biotechnology* 35 (11 avril 2017): 316- 19. https://doi.org/10.1038/nbt.3820.
11. Goodstadt, Leo. « Ruffus: A Lightweight Python Library for Computational Pipelines ». *Bioinformatics (Oxford, England)* 26 (2010): 2778- 79. https://doi.org/10.1093/bioinformatics/btq524.
12. « H2020 INVITE Project | INnovations in Plant VarIety Testing in Europe ». Consulté le 12 août 2022. https://www.h2020-invite.eu/.

13. Hsi-Yang Fritz, Markus, Rasko Leinonen, Guy Cochrane, et Ewan Birney. « Efficient storage of high throughput DNA sequencing data using reference-based compression ». *Genome Research* 21, nº 5 (2011): 734- 40. https://doi.org/10.1101/gr.114819.110.
14. « Interval Bio | Allegro(R) Bioinformatics ». Consulté le 12 août 2022. https://interval.bio/allegro_bioinformatics.html.
15. Jette M, Grondona M, et Yoo A. « Slurm: Simple Linux Utility for Resource Management », 2003. https://slurm.schedmd.com/slurm_design.pdf.
16. Köster, Johannes, et Sven Rahmann. « Snakemake—a scalable bioinformatics workflow engine ». *Bioinformatics* 28, nº 19 (2012): 2520- 22. https://doi.org/10.1093/bioinformatics/bts480.
17. Leipzig, Jeremy. « A review of bioinformatic pipeline frameworks ». *Briefings in Bioinformatics* 18, nº 3 (1 mai 2017): 530- 36. https://doi.org/10.1093/bib/bbw020.
18. Li, Heng, et Richard Durbin. « Fast and accurate short read alignment with Burrows–Wheeler transform ». *Bioinformatics* 25, nº 14 (2009): 1754- 60. https://doi.org/10.1093/bioinformatics/btp324.
19. Li, Heng, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, et Richard Durbin. « The Sequence Alignment/Map format and SAMtools ». *Bioinformatics* 25, nº 16 (2009): 2078- 79. https://doi.org/10.1093/bioinformatics/btp352.
20. McKenna, Aaron, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, et al. « The Genome Analysis Toolkit: A MapReduce Framework for Analyzing next-Generation DNA Sequencing Data ». *Genome Research* 20, nº 9 (2010): 1297- 03. https://doi.org/10.1101/gr.107524.110.
21. Mölder, Felix, Kim Philipp Jablonski, Brice Letcher, Michael B. Hall, Christopher H. Tomkins-Tinch, Vanessa Sochat, Jan Forster, et al. « Sustainable Data Analysis with Snakemake ». F1000Research, 19 avril 2021. https://doi.org/10.12688/f1000research.29032.2.
22. Muzzey, Dale, Eric A. Evans, et Caroline Lieber. « Understanding the Basics of NGS: From Mechanism to Variant Calling ». *Current Genetic Medicine Reports* 3, nº 4 (2015): 158- 65. https://doi.org/10.1007/s40142-015-0076-8.
23. Ott, Alina, Sanzhen Liu, James C. Schnable, Cheng-Ting 'Eddy' Yeh, Kai-Sin Wang, et Patrick S. Schnable. « tGBS® genotyping-by-sequencing enables reliable genotyping of heterozygous loci ». *Nucleic Acids Research* 45, nº 21 (2017): e178. https://doi.org/10.1093/nar/gkx853.
24. « RésiLens - Constitution et évaluation d'une collection de ressources génétiques de lentille (*Lens culinaris* Medik.). Recherche de sources de résistance aux bruches et aux pourritures racinaires incluant *Aphanomyces euteiches* - Accueil ». Consulté le 12 août 2022. https://www6.inrae.fr/ResiLens/.

25. « Running a Job on HPC using Slurm | HPC | USC », 6 mars 2019. https://web.archive.org/web/20190306044130/https://hpcc.usc.edu/support/documentation/slurm/.
26. « samtools-index(1) manual page ». Consulté le 10 août 2022. https://www.htslib.org/doc/samtools-index.html.
27. Scaglione, Davide, Sara Pinosio, Fabio Marroni, Eleonora Di Centa, Alice Fornasiero, Gabriele Magris, Simone Scalabrin, Federica Cattonaro, Gail Taylor, et Michele Morgante. « Single primer enrichment technology as a tool for massive genotyping: a benchmark on black poplar and maize ». *Annals of Botany* 124, nº 4 (2019): 543- 51. https://doi.org/10.1093/aob/mcz054.
28. « TOML: English v1.0.0 ». Consulté le 12 août 2022. https://toml.io/en/v1.0.0.
29. Torkamaneh, Davoud, Jérôme Laroche, Maxime Bastien, Amina Abed, et François Belzile. « Fast-GBS: a new pipeline for the efficient and highly accurate calling of SNPs from genotyping-by-sequencing data ». *BMC Bioinformatics* 18 (2017): 5. https://doi.org/10.1186/s12859-016-1431-9.
30. « YAML Ain't Markup Language (YAML™) revision 1.2.2 ». Consulté le 10 août 2022. https://yaml.org/spec/1.2.2/.

# Annexe 1 : Présentation de la structure d'accueil

Créé le 1er janvier 2020, INRAE, l'Institut National de Recherche pour l'Agriculture, l'Alimentation et Environnement, est le résultat de la fusion entre l'INRA, l'Institut national de la recherche agronomique et Irstea, l'Institut national de recherche en sciences et technologies pour l'environnement et l'agriculture. Il s'agit du premier organisme de recherche à se spécialiser conjointement dans les trois domaines de l'alimentation, l'agriculture et l'environnement. Placé sous la tutelle du ministère de l'Agriculture et de la souveraineté alimentaire, INRAE se concentre sur des projets dont les enjeux sont territoriaux comme la santé des animaux d'élevage en lien à la santé humaine, la transformation de l'agriculture, des systèmes socio-écologiques et alimentaires, mais aussi sur des enjeux plus globaux comme la structure, le fonctionnement et l'évolution des écosystèmes continentaux faiblement anthropisés ou la compréhension des grandes fonctions du végétal. L'institut emploie plus de 10 000 agents dans 268 unités de recherches, de service et d'expérimentations réparties dans 18 centres à travers toute la France.

Mon stage a été réalisé à l'EPGV (US 1279), le laboratoire d'Étude du Polymorphisme des Génomes Végétaux, une plateforme de génomique du centre INRAE Île-de-France - Versailles-Grignon, rattachée à l'université de Paris-Saclay, et localisée sur le site du CEA/Centre National de Séquençage à Évry. Il s'agit d'une des quatre plateformes de l'Infrastructure de recherche distribuée INRAE Genomics. L'EPGV est spécialisée dans la détection et l'analyse de polymorphismes dans les génomes végétaux, grâce notamment aux techniques de génotypage ciblé par séquençage. Elle est impliquée dans de nombreux projets nationaux et internationaux de grande ampleur, tout en ayant à cœur de répondre aux projets plus spécifiques des unités de recherche. Composée de six permanents et accueillant actuellement une ingénieure d'étude en CDD, l'unité est répartie en 2 équipes "Production de données et développement technologique" et "Informatique et bioinformatique". L'unité a accès aux dernières technologies de séquençage génomique, incluant les séquenceurs courtes lectures (Illumina et MGI) et longues lectures (ONT Nanopore) pour la génération de séquences à haut débit, mais aussi les outils permettant de générer et analyser les cartes optiques (Bionano Saphyr).

# Annexe 2 : Bilan personnel du stage

Ce stage a été pour moi l'occasion d'appliquer mes compétences en programmation pour un projet complet. J'ai pu en apprendre plus sur le temps de planification nécessaire avant de commencer le développement à proprement parler, et comprendre les implications de travailler avec les contraintes d'un cluster de calcul partagé. Avec des ressources limitées, il devient nécessaire de prendre en compte la présence des autres utilisateurs et de choisir les paramètres les plus adaptés. J'ai également eu l'occasion assez tôt de mettre en application mon pipeline sur les données de projets réels sans pression de rendu, ce qui a permis d'identifier et de corriger des limites majeures suffisamment tôt dans le développement.

Étant en autonomie et libre sur le choix des outils employés pour l'automatisation, j'ai recherché les différentes solutions afin de les comparer, ce qui m'a permis d'avoir une vue d'ensemble sur les avantages et inconvénients des frameworks de pipeline les plus courants. Ensuite, j'ai pu appliquer des technologies qui me sont familières et expérimenter de nouvelles méthodes qui m'ont permis de me perfectionner à la fois en python, mais surtout en maîtrise de BASH. J'ai également pris la pleine conscience de la puissance d'AWK pour la lecture de fichiers volumineux.

J'ai été ravi de découvrir dans l'unité des valeurs proches des miennes, qu'il s'agisse de sobriété des programmes, du partage des connaissances ou de la rigueur des analyses, ce qui nous a permis d'avoir des objectifs communs dès le début du stage.

Les multiples présentations auxquelles j'ai pu assister au Génoscope ou à INRAE m'ont fait découvrir des sujets de recherche aussi variés que les moyens, comme le développement d'un algorithme de prédiction d'interaction entre protéines ou encore la découverte d'une ADN polymérase favorisant une base non canonique.

J'ai été pleinement intégré à l'unité, participant aux réunions bio-informatiques et générales, devant faire des comptes-rendus réguliers. En particulier à l'attention du personnel non-bioinformatique, ce qui impliquait un certain degré de vulgarisation et une communication claire, que j'ai développée.

Finalement, ce stage m'a permis de développer des réflexes méthodologiques, notamment l'habitude de noter ce qui a été fait et ce qu'il reste à faire à la fin de chaque journée, ce qui me permettait d'être plus efficace le lendemain. Il m'a également donné une meilleure idée du milieu de la bioinformatique actuelle et de ses enjeux, notamment en termes d'optimisation.

# Résumé

**Titre :** Développement d'un pipeline modulaire pour un cluster SLURM partagé, adapté à l'analyse du polymorphisme des génomes : Application au tGBS chez les végétaux

**Mots clefs :** Automatisation, appel de variant, HPC, STAMPS, accessibilité

L'émergence des nouvelles technologies de séquençage entraîne un développement des pipelines d'analyses bioinformatiques nécessaires pour traiter ce flot de données génomiques. Les cadres de pipelines permettent une automatisation des processus, souvent au détriment de l'accessibilité ou de la flexibilité des analyses, et ne sont pas adaptés à toutes les situations. Il s'agit dans le cadre de ce stage d'automatiser un pipeline d'alignement et d'appel de variants sur des données de génotypage ciblé par séquençage (tGBS), à destination d'une plateforme de séquençage dans un environnement de cluster SLURM partagé. Simple Targeted GBS Automated and Modular Pipeline for SLURM (STAMPS) est un pipeline d'analyse écrit en python et BASH pensé pour être en adéquation avec les contraintes et ressources informatiques disponibles, tout en optimisant leur usage. Il se veut être un cadre modulaire et flexible brillant dans une situation où les frameworks classiques se retrouvent limités. Avec une formation minimale, il peut être rapidement pris en main par des non-bioinformaticiens sans compétences informatiques avancées. Bien qu'il ait été créé pour réaliser de l'appel de variants sur des données de tGBS, ce STAMPS se veut être généraliste et adaptable à tout type d'analyse.

# Abstract

**Title:** Development of a modular pipeline for a shared SLURM cluster, suited for genome polymorphism analysis: application on plant tGBS.

**Keywords:** Automation, variant calling, HPC, STAMPS, accessibility

Recent upcoming of new sequencing technologies led to the development of bioinformatics analysis pipelines required to handle this flow of genomic data. Pipeline frameworks allow for automation of these processes, often at the cost of accessibility or analysis flexibility, and are not always adapted. This internship aimed to automate an alignment and variant-calling pipeline for targeted genotyping-by-sequencing (tGBS) data developed by a sequencing facility, in a shared SLURM cluster environment. Simple Targeted GBS Automated and Modular Pipeline for SLURM (STAMPS) is an analysis pipeline written in python and BASH, designed to optimize the usage of the computing resources in conformation with the constraints. It aims to be a modular and flexible framework, shining where the commonly available frameworks are limited. With only a short training, non-bioinformaticians can quickly get used to it, even without advanced informatic skills. Despite being originally designed as a variant calling pipeline for tGBS data, STAMPS is a generalist framework, adaptable to any type of analysis.