Pipelines d'Analyse Bioinformatiques : solutions offertes par les Systèmes de Workflows, Cadre de représentation et Étude de la Réutilisation

Marine Djaffardjy

université PARIS-SACLAY

Pipelines d'Analyse Bioinformatiques : solutions offertes par les Systèmes de Workflows, Cadre de représentation et Étude de la Réutilisation

Bioinformatics Data Analysis Pipelines: Solutions Provided by Workflow Systems, Representation Framework, and Study of Reusability

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580 : sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique
Graduate School : Informatique et Sciences du Numérique Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche **Laboratoire Interdisciplinaire des Sciences du Numérique (LISN) (Université Paris-Saclay, CNRS)** sous la direction de **Sarah COHEN-BOULAKIA**, Professeure (Université Paris-Saclay, LISN) et le co-encadrement de **Alban GAIGNARD**, Ingénieur de recherche (Université de Nantes, ITX)

Thèse soutenue à Paris-Saclay, le 23 octobre 2023, par

Marine DJAFFARDJY

Composition du jury

Membres du jury avec voix délibérative

Caroline APPERT Directrice de Recherche, CNRS, LISN, Université Paris-Saclay	Présidente
Olivier DAMERON Professeur, IRISA, Université de Rennes	Rapporteur & Examinateur
Pierre POULAIN Maître de conférences HDR, IJM, Université Paris Cité	Rapporteur & Examinateur
Bruno CREMILLEUX Professeur, GREYC, Université de Caen	Examinateur
Marie-Dominique DEVIGNES Chargée de Recherche, CNRS, LORIA, Université de Lorraine	Examinatrice

THESE DE DOCTORAT
NNT : 2023UPASG059

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Pipelines d'Analyse Bioinformatiques : solutions offertes par les Systèmes de Workflows, Cadre de représentation et Étude de la Réutilisation

Mots clés : Workflows scientifiques, Réutilisation et échange de pipelines, Intégration de données biologiques, Analyse de données multi-échelles

Résumé : La bioinformatique est un domaine multidisciplinaire qui combine biologie, informatique et statistiques, permettant de mieux comprendre les mécanismes du vivant.
Son fondement repose essentiellement sur l'analyse des données biologiques. L'émergence de nouvelles technologies, en particulier les avancées majeures dans le domaine du séquençage, a entraîné une croissance exponentielle des données, posant de nouveaux défis en matière d'analyse et de gestion des données.
Pour exploiter ces données, des pipelines sont utilisés, enchaînant des outils et des processus informatiques pour conduire les analyses de manière fiable et efficace. Cependant, la crise de la reproductibilité dans la recherche scientifique souligne la nécessité de rendre les analyses reproductibles et réutilisables par des tiers.

Les systèmes de workflows scientifiques ont émergé comme une solution pour rendre les pipelines plus structurés, compréhensibles et reproductibles. Les workflows décrivent des procédures en plusieurs étapes coordonnant des tâches et leurs dépendances de données. Ces systèmes aident les bioinformaticiens à concevoir et exécuter des workflows, et facilitent leur partage et réutilisation. En bioinformatique, les systèmes de workflows les plus populaires sont Galaxy, Snakemake, et Nextflow.

Cependant, la réutilisation des workflows fait face à des difficultés, notamment l'hétérogénéité des systèmes de workflows, le manque d'accessibilité des workflows et le besoin de bases de données publiques de workflows. De plus, l'indexation et le développement de moteurs de recherche de workflows sont nécessaires pour faciliter la recherche et la réutilisation des workflows.

Dans un premier temps, nous avons développé une méthode d'analyse des spécifications de workflows afin d'extraire plusieurs caractéristiques représentatives à partir d'un ensemble de données de workflows. Notre objectif était de proposer un cadre standard pour leur représentation, indépendamment de leur langage de spécification.
Dans un second temps, nous avons sélectionné un ensemble de caractéristiques de ces workflows et les avons indexées dans une base de données relationnelle, puis dans un format structuré sémantique.
Enfin, nous avons mis en place une approche pour détecter les similarités entre les workflows et les processeurs, permettant ainsi d'observer les pratiques de réutilisation adoptées par les développeurs de workflows.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title: Bioinformatics Data Analysis Pipelines: Solutions Provided by Workflow Systems, Representation Framework, and Study of Reusability

Keywords: Scientific workflows, Reuse and protocol exchange, Biological data integration, Data Analysis

Abstract : Bioinformatics is a multidisciplinary field that combines biology, computer science, and statistics, aiming to gain a better understanding of living mechanisms. It relies primarily on the analysis of biological data. Major technological improvements, especially sequencing technologies, gave rise to an exponential increase of data, laying out new challenges in data analysis and management.

In order to analyze this data, bioinformaticians use pipelines, which chain computational tools and processes. However, the reproducibility crisis in scientific research highlights the necessity of making analyses reproducible and reusable by others.

Scientific workflow systems have emerged as a solution to make pipelines more structured, understandable, and reproducible. Workflows describe procedures with multiple coordinated steps involving tasks and their data dependencies. These systems assist bioinformaticians in designing and executing workflows, facilitating their sharing and reuse. In bioinformatics, the most popular workflow systems are Galaxy, Snakemake, and Nextflow.

However, the reuse of workflows faces challenges, including the heterogeneity of workflow systems, limited accessibility to workflows, and the need for public workflow databases. Additionally, indexing and developing workflow search engines are necessary to facilitate workflow discovery and reuse.

In this study, we developed an analysis method for workflow specifications to extract several representative characteristics from a dataset of workflows. The goal was to propose a standardized representation framework independent of the specification language. Additionally, we selected a set of workflow characteristics and indexed them into a relational database and a structured semantic format. Finally, we established an approach to detect similarity between workflows and between processors, enabling us to observe the reuse practices adopted by workflow developers.

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué de près ou de loin à la réalisation de cette thèse. Leur soutien indéfectible, leurs conseils éclairés et leur engagement ont largement contribué à la réussite de ce travail de recherche.

En premier lieu, je tiens à exprimer ma reconnaissance envers mes encadrants, Sarah et Alban, dont la guidance experte et l'encouragement constant ont été des éléments déterminants. Leur expertise, leur disponibilité et leur capacité à susciter la réflexion ont grandement enrichi mon parcours académique. J'ai également eu la chance de profiter de leurs qualités humaines exceptionnelles, et ai énormément appris sur le travail d'équipe à leur contact.

Je tiens à exprimer ma reconnaissance envers les stagiaires qui ont participé à cette recherche, en particulier George Marchment, Clémence Sèbe, Noémie Bossut et Maxime , qui ont énormément apporté dans mes travaux. Merci à eux pour leur motivation et leur persévérance.

Mes remerciements s'adressent également à toutes les personnes qui ont accepté de participer à mes interviews, Raphaël Blanchet, Pierre Lindenbaum, Éric Charpentier et Frédéric Lemoine. Leur générosité intellectuelle et leur volonté de partager leurs connaissances ont été essentielles.

Mes pensées vont également à tous ceux qui m'ont aidé et conseillé dans ce travail de thèse, notamment Khalid Belhajjame et (encore !) Frédéric Lemoine, avec qui j'ai eu la chance de collaborer pour des publications, et qui ont partagé leurs connaissances, leur temps, et leur patience. Merci également à Joe Raad qui a pris de son temps pour échanger avec moi et me conseiller sur des sujets très intéressants.

J'adresse ma gratitude à mes rapporteurs Olivier Dameron et Pierre Poulain, à Caroline Appert qui a présidé le jury et à mes examinateur et examinatrice Bruno Crémilleux et Marie-Dominique Devignes. Je les remercie chaleureusement d'avoir accepté d'évaluer mes travaux et m'avoir apporté leurs remarques et leur bienveillance.

L'atmosphère au sein de l'équipe bioinformatique s'est révélée particulièrement conviviale. Je tiens à exprimer ma gratitude envers toutes les personnes avec qui j'ai partagé un bureau, et des moments de pause plus qu'agréables ! Je souhaite adresser des remerciements chaleureux à Fanny Pouyet, avec qui j'ai eu le privilège de collaborer de nombreuses fois dans le cadre de l'enseignement.

Je tiens à exprimer ma reconnaissance envers mes collègues de l'Institut du Thorax pour leur sympathie lors de nos interactions et rencontres.

Enfin, mes remerciements vont à Alain Denise, Nicolas Thiéry, Ouriel Grynszpan, et tous les autres enseignants avec qui j'ai eu le plaisir de travailler au cours de ces trois années. Votre collaboration a été des plus agréables.

Enfin, je souhaite exprimer ma gratitude envers ma famille et mes amis qui m'ont soutenu tout au long de ce parcours académique. Leur soutien moral et émotionnel a été une force motrice qui m'a permis de surmonter les défis et d'atteindre mes objectifs. Merci en particulier à ma mère pour ses relectures de dernière minute, et à Léo pour son soutien sans faille, sans lequel cette période de rédaction aurait été bien plus difficile, et ses superbes schémas plus propres que les miens. Merci à Lucas pour son aide précieuse, surtout pour la dernière ligne droite !

Ce travail de thèse n'aurait pas été possible sans la contribution précieuse de chacun de vous. Merci du fond du cœur pour votre collaboration, votre expertise et votre amitié.

Table des matières

1 - Introduction

La bioinformatique est un domaine multidisciplinaire qui combine biologie, informatique et statistiques et offre de nouvelles opportunités pour mieux comprendre les nombreux mécanismes du vivant. L'analyse de données biologiques est un élément essentiel de nombreux champs d'application de la bioinformatique.

L'avènement de nouvelles technologies, comme les nouvelles technologies de séquençage [1] Illumina Novaseq ou Nanopore Sequencing, sont à l'origine d'une croissance exponentielle des données disponibles.

Pour les exploiter, de nombreuses étapes de pré-traitement sont nécessaires pour transformer et filtrer les données (à titre d'exemple, les technologies de séquençage sont sujettes aux erreurs, et nécessitent des algorithmes capables de gérer ces erreurs et incertitudes).

Un grand nombre de ces étapes de pré-traitement impliquent des algorithmes complexes, et de plus en plus d'outils informatiques sont développés pour les réaliser. La majorité de ces outils sont spécifiquement conçus pour effectuer un traitement particulier. Un exemple d'outil couramment utilisé en bioinformatique, notamment en génomique, est bowtie [2], qui est utilisé pour l'alignement de séquences d'ADN ou d'ARN sur un génome de référence. L'alignement des séquences est une étape nécessaire à de nombreux types d'analyses génomiques (analyse de données RNA-Seq, études de variations génétiques, *etc.*), et peut être longue et coûteuse en ressources lorsqu'elle est appliquée à un volume de données important.

Les analyses de données impliquent donc plusieurs outils et étapes. Les exécuter manuellement est long et propice aux erreurs, les bioinformaticiens ont donc cherché à automatiser leur orchestration pour les rendre plus efficaces. **Les pipelines** permettent de coordonner ces outils et scripts. Un pipeline est un ensemble d'opérations ou de tâches interconnectées, effectuées de manière séquentielle ou parallèle. Dans le contexte de la bioinformatique, un pipeline est une séquence ordonnée d'outils et de processus informatiques utilisée pour analyser et interpréter des données biologiques.
Les résultats d'une étape d'analyse sont utilisés comme entrée pour l'étape suivante. Les pipelines bioinformatiques peuvent effectuer des tâches en parallèle, de telle sorte que des ensembles de données indépendants sont traités en même temps

afin d'accélérer le processus d'analyse.
L'objectif principal d'un pipeline bioinformatique est donc de permettre aux chercheurs et aux scientifiques d'organiser et d'effectuer des analyses de données de manière systématique, en réduisant les erreurs humaines et en améliorant l'efficacité des calculs et des traitements.
Les pipelines doivent être extensibles pour ajouter de nouveaux outils, pouvoir gérer des étapes séquentielles et parallèles, des dépendances complexes, ainsi que l'intégration et l'orchestration de divers outils et scripts avec leurs paramètres respectifs [3].

Face à la quantité croissante d'outils, de scripts et de pipelines utilisés en bioinformatique, et de façon générale dans d'autres domaines de la recherche scientifique, une préoccupation majeure a émergé ces dernières années : **la crise de la reproductibilité** [4]. En effet, la complexité des analyses de données, la diversité et le nombre de logiciels utilisés ainsi que l'hétérogénéité des environnements et plateformes rendent la reproduction des résultats d'une étude ou d'une analyse difficile, même par leurs propres auteurs.
Cette crise remet en question la fiabilité des découvertes scientifiques et souligne la nécessité de mettre en place des pratiques et des outils permettant de garantir la **reproductibilité** des analyses scientifiques.
On attend notamment des scientifiques qu'ils partagent leurs ressources, y compris les ensembles de données, les outils d'analyse et les pipelines.

Étant donné que les analyses sont complexes, l'implémentation de tels pipelines peuvent être des tâches chronophages et difficiles pour les bioinformaticiens. Cependant, toutes les analyses ne sont pas entièrement uniques.
Les données biologiques sont hétérogènes, mais il n'existe pas un nombre infini de méthodes pour les générer. Peu de nouvelles méthodes sont développées, et lorsqu'une méthode devient établie, elle est applicable à un grand nombre de cas d'utilisation [5]. Par exemple, *la spectrométrie de masse est incontournable en protéomique*. On peut également citerles *microarray*, qui permettent de mesurer les niveaux d'expression des transcrits d'un échantillon, ont été largement utilisées pendant de nombreuses années depuis leur apparition (une grande partie des maladies ont été étudiées à l'aide de *microarray*). Si toutes les puces ne sont pas identiques, il existe des normes spécifiques pour chaque type de puce. Les scientifiques peuvent donc être amenés à vouloir conduire des analyses sur des données nouvelles, qui ont déjà été implémentées pour des données générées par les mêmes puces.

En outre, certaines étapes de traitement peuvent être partagées entre plusieurs types d'analyses. Par exemple, certaines étapes de pré-traitement sont courantes en bioinformatique, comme l'alignement de séquences, le filtrage de données,...
La possibilité de **réutiliser** des pipelines est donc un atout majeur pour faciliter l'implémentation de nouveaux pipelines et la réalisation de nouvelles analyses, et d'exploiter l'expertise d'autres utilisateurs. La **réutilisation** de pipelines déjà établis et validés permet d'améliorer la qualité et la fiabilité des pipelines.
La réutilisabilité d'un pipeline permet également de partager et d'évaluer des méthodes d'analyses et de comparer leurs résultats.
Pour qu'un pipeline soit réutilisable, il doit être reproductible, mais cela ne suffit pas.

L'approche initiale d'implémentation des pipelines, qui se basait sur des scripts, a rapidement révélé ses limites en termes de reproductibilité et de réutilisabilité [6] : un pipeline de taille modérée peut rapidement devenir difficile à comprendre et à maintenir, et encore plus difficile à réutiliser par des tiers. En outre, une telle approche ne permet pas l'implémentation de pipelines d'envergure manipulant de grandes quantités de données [7, 8].

Les systèmes de workflows scientifiques ont émergé comme une solution prometteuse à ces problèmes. Ils offrent un approche structurée, standardisée et reproductible pour l'implémentation de pipelines. Ils facilitent la compréhension des pipelines, ainsi que leur partage et leur réutilisation [9].
Un workflow est une description précise d'une procédure en plusieurs étapes visant à coordonner plusieurs tâches et leurs dépendances de données. Chaque tâche représente l'exécution d'un processus informatique, tel que l'invocation d'un service, l'appel à un outil en ligne de commande, l'accès à une base de données ou encore l'exécution d'un script ou d'un flux de traitement de données [10]. Un workflow peut donc être vu comme une chaîne de *processeurs* (ou modules), chacun effectuant une tâche. Les processeurs peuvent encapsuler des outils ou des scripts. Ces processeurs sont chaînés par flux de données : l'entrée d'un processeur est connectée à la sortie du précédent, ce qui détermine l'ordre dans lequel ils sont exécutés.

Les systèmes de gestion de workflows scientifiques [11] ont donc été conçus pour aider les bioinformaticiens à concevoir et à exécuter des workflows 1.1, à plusieurs niveaux [12] tout au long du cycle de vie des pipelines. Des systèmes de gestion de workflows populaires en bioinformatique sont Galaxy [13], Snakemake [14] et Nextflow [15]. Taverna [16] fut le système pionnier ; il a cessé d'être maintenu en 2016. Les workflows sont de plus en plus adoptés par les scientifiques pour l'im-

plémentation de pipelines d'analyses de données.

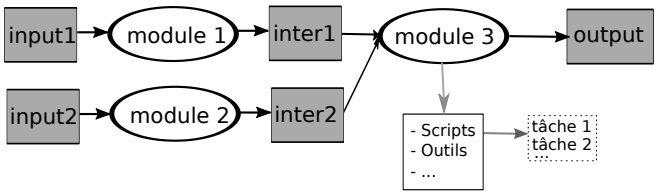

Figure 1.1 – **Workflows** : catégorie d'implémentation de **pipelines**. Ils sont constitués de **modules** qui encapsulent des processus informatiques (qui effectuent des **tâches**) reliés par leurs inputs et outputs (les données, input/intermédiaire/output sont représentés par des carrés gris)

Malgré ces éléments de solutions, la réutilisation des workflows se heurte à trois difficultés principales. Premièrement, il existe une grande variété de systèmes de workflows, ce qui implique une grande hétérogénéité. Il y a un besoin fort de standardiser ces workflows.
Deuxièmement, les workflows ne sont pas toujours accessibles, ce qui est causé par le manque de bases de données publiques de workflows d'envergure, la majorité des workflows n'étant pas disponibles publiquement. Construire de telles bases est donc important.
Enfin, il est important de mettre en place des fonctionnalités au sein de ces bases de données, ou plateformes de partage, pour permettre aux utilisateurs de retrouver des workflows adaptés à leurs besoins.
Pour mettre en place des fonctionnalités qui favorisent efficacement la réutilisation des workflows, il est essentiel de mieux comprendre comment le partage et la réutilisation de ces méthodes sont effectués. L'étude des pratiques des utilisateurs joue un rôle fondamental pour identifier leurs besoins, établir des bonnes pratiques et évaluer l'impact des workflows réutilisés.

En d'autres termes, nous nous intéressons à trois questions de recherche :

- **I**) Représentation uniforme des workflows ;
- **II**) Regroupement des informations des workflows dans une base de données ;
- **III**) Identification de la réutilisation dans un ensemble de workflows.

Ces questions de recherche sont particulièrement présentes dans le contexte plus spécifique de cette thèse : le projet PRIME du CNRS intitulé R2-P2. R2-P2 regroupe des bioinformaticiens, biologistes, médecins et informaticiens issus de 5 partenaires – le LISN, l'Institut du Thorax (ITX) de Nantes, l'Institut Pasteur, le LAMSADE – qui collaborent autour de la problématique générale de la réutilisation de pipelines d'analyse de données mis en oeuvre pour l'étude des anévrismes

intracrâniens.
Dans ce contexte, l'ITX produit des pipelines d'analyse de données pour tendre vers une meilleure compréhension de la présence, du développement et éventuellement de la rupture des anévrismes intracrâniens. Les données sur lesquelles portent leurs analyses sont fortement hétérogènes et multi-modales (des données d'imagerie aux données génomiques).

L'objectif du projet R2-P2 est de tendre vers la conception d'un cadre pour le développement et l'exécution de protocoles d'analyse de données reproductibles et réutilisables pour l'étude des anévrismes intracrâniens.

Cette thèse a pour but de favoriser la réutilisation et l'indexation des workflows.
Elle propose quatre contributions.
La première est l'introduction d'une méthode d'analyse des spécifications de workflows. Nous identifions des caractéristiques représentatives des workflows, ou *annotations*, que nous extrayons des spécifications. Cette méthode fait également appel à des sources externes à la spécification pour compléter ces annotations. Nous avons introduit une représentation structurée et standard de ces workflows, indépendante de leur langage de spécification.

La deuxième contribution est l'établissement d'un riche jeu de données structuré et uniforme de workflows annotés. Nous avons constitué ce jeu de données à partir de fichiers de spécification de workflows, que nous avons analysé à l'aide d'une nouvelle méthode d'annotation. Ce jeu de donnée est proposé à la fois sous la forme d'une base de données relationnelle et d'une base de connaissances RDF.

La troisième contribution est une étude du partage et de la réutilisation au sein des workflows. Nous avons introduit et implémenté des méthodes de détection de la réutilisation et fait état de pratiques d'utilisateurs.

La quatrième contribution est le recensement de besoins de développeurs de workflows. Des interviews ont été conduites sur quatre développeurs de workflows du projet R2-P2 pour identifier les besoins des utilisateurs lors de la conception, l'implémentation, l'exécution et le partage de pipelines d'analyses de données. Ces besoins utilisateurs ont été catégorisés. Pour chacune de ces catégories de besoins, un état de l'art précis des solutions a été dressé.

Plus particulièrement, les systèmes de gestion de workflows que nous étudions sont

Snakemake et Nextflow, deux des systèmes les plus largement adoptés par les bioinformaticiens ayant des connaissances en programmation. L'article *Developing and reusing bioinformatics data analysis pipelines using scientific workflow systems* [17], publié dans le journal *Computational and Structural Biotechnology Journal* et écrit par un ensemble de co-auteurs du projet R2-P2 est la publication majeure de ces travaux de thèse.
Les développements conséquents associés à nos analyses sont regroupés dans les dépots gits suivants :

- Annotation de workflows Snakemake : `https://github.com/mdjaffardjy/Snakemake_workflow_analysis` ;
- Annotation de workflows Nextflow : `https://github.com/mdjaffardjy/AnalyseDonneesNextflow` ;
- Étude des pratiques de réutilisation : `https://github.com/mdjaffardjy/Reuse_in_processes`.

J'ai conçu et implémenté la méthode d'analyse de spécification sur le système Snakemake et co-encadré deux stages qui ont adapté cette méthode au système Nextflow. Le schéma de la base de données relationnelle et son remplissage a été effectué lors d'un troisième stage que j'ai co-encadré. J'ai également implémenté les méthodes de détection de la réutilisation.

La suite de ce manuscrit est organisée comme suit.

Le chapitre 2 met en évidence les besoins des utilisateurs de workflows. Dans ce chapitre, nous présentons un résumé détaillé des entretiens réalisés avec quatre développeurs de workflows du projet R2-P2. Des besoins identifiés dans ces entretiens, nous identifions une typologie d'utilisateurs et listons des cas d'utilisation.

Le chapitre 3 présente un état de l'art des solutions répondant à ces besoins. Dans ce chapitre, les solutions disponibles pour l'implémentation de pipelines sont présentées et comparées. Les besoins utilisateurs identifiés du chapitre 2 sont rattachés à différentes étapes du processus d'analyse de données avec des pipelines. Un état de l'art des solutions disponibles aux problèmes propres à chacune de ces étapes est établi. Dans un premier temps, des solutions "génériques", indépendantes de l'implémentation des pipelines sont décrites. Dans un second temps, les solutions spécifiques aux workflows scientifiques sont présentées.

Le chapitre 4 présente les résultats de l'étude que nous avons réalisée sur la réutilisa-

tion de workflows scientifiques. Dans ce chapitre, la constitution du jeu de données, la méthode d'annotation ainsi que la base de données et base de connaissances résultantes sont décrites. La démarche de l'étude sur la réutilisation est ensuite présentée. Tout d'abord, les pratiques de partage des utilisateurs sont analysées. Trois niveaux de réutilisation sont ensuite étudiés : la réutilisation de workflows entiers, la réutilisation des processeurs et la réutilisation des outils bioinformatiques. Pour chacun de ces niveaux, des pratiques de réutilisation sont décrites.

Le chapitre 5 reprend l'ensemble des contributions de ces travaux de thèse, discute nos résultats et propose un ensemble de perspectives à ces travaux.

2 - Besoins utilisateurs

Sommaire

2.1 . Introduction

Dans ce chapitre nous proposons de poser un état des lieux des besoins des concepteurs, développeurs et utilisateurs de pipelines d'analyse de données, recensés dans la communauté bioinformatique. Plus précisément, ce chapitre rend compte d'une série d'interviews que nous avons conduites auprès de bioinformaticiens collaborateurs, dans le contexte du projet R2-P2 et donc lié à la fois au projet ICAN et à notre collaboration avec le Hub de bioinformatique de l'Institut Pasteur. Bien que restreint, ce groupe de personnes est impliqué dans des projets nationaux et internationaux d'envergure, ils ont des tâches quotidiennes et des rôles qui nous ont semblé particulièrement représentatifs des activités menées dans des projets bioinformatiques dans lesquels l'analyse de données bioinformatique est centrale.

La mise en place d'un pipeline d'analyse de données est une entreprise complexe qui nécessite une compréhension approfondie des données, le choix judicieux des outils appropriés pour l'analyse, ainsi qu'une maîtrise de leur orchestration. Dans le prochain chapitre (Chapitre 3.2), nous examinerons en détail les diverses méthodes d'implémentation de ces pipelines. Ils peuvent être mis en œuvre sous forme de scripts, de notebooks (tels que Jupyter Notebook) ou de workflows (comme Snakemake, Nextflow, etc.). Chacune de ces approches d'implémentation offre des fonctionnalités conçues pour simplifier le développement pour les utilisateurs de pipelines. Parmi ces fonctionnalités essentielles, on compte la gestion des dépendances entre les tâches ainsi que des dépendances logicielles. Les systèmes de gestion de workflows sont hautement modulaires, chaque module étant une entité autonome et indépendante.
Ces solutions offrent une variété de modes d'implémentation, tels que des interfaces graphiques ou des langages spécifiques, qui s'adressent aux utilisateurs novices ou expérimentés.

L'objectif de ce chapitre est de mettre en évidence différents profils d'utilisateurs, comprendre les motivations de ces utilisateurs, leur démarche et identifier leurs besoins. Un second objectif intrinsèquement lié est relatif à une typologie de besoins, décrits sous la forme de cas d'utilisation concrets, en lien direct avec les interviews. Ces cas d'utilisation serviront de motivation à la présentation des solutions existantes dans le chapitre suivant.

Ce chapitre est organisé comme suit. Nous décrivons d'abord notre démarche dans la gestion des interviews et le contexte général de cette collecte de besoins (2.2). Nous proposons ensuite une présentation synthétique des interviews (2.3). Nous identifions en 2.4 un ensemble de profils d'utilisateurs ainsi que des cas d'utilisation, illustrant le panel de besoins des bioinformaticiens auxquels il est nécessaire de pouvoir répondre. Nous allons ensuite (2.5) présenter le "cycle de vie des pipelines" selon lequel nous catégoriserons ces besoins des utilisateurs. Enfin, nous concluons (2.6).

2.2 . Démarche et contexte de collecte des besoins

2.2.1 . Démarche

Les interviews ont été réalisées comme des entretiens semi-directifs, durant lesquels les participants ont répondu à une base commune de questions durant une heure. Le point de départ des entretiens était la description d'un ou deux pipelines que les participants avaient produits. A partir de cette description, les participants étaient

amenés à répondre à des questions relatives à leurs pratiques de développement de pipelines, en prenant appui sur la description de pipelines précis puis en généralisant à leur pratiques d'analyses de données génomiques habituelles.
Les participants ont donné leur autorisation explicite pour être cités dans les compte-rendus d'interview.
Les questions étaient structurées autour des thèmes suivants :

- Contexte d'écriture du pipeline : avec qui est il écrit, à qui le pipeline est destiné, *etc.* ;
- Données d'analyse : type, volume, nature (sensibles ou non), *etc.* ;
- Déploiement du pipeline ;
- Analyse des résultats ;
- Pérennité du pipeline ;
- Partage du pipeline ;
- Réutilisation de pipelines.

2.2.2 . Contexte principal de la collecte : le projet ICAN

Pour illustrer les besoins des bioinformaticiens dans le développement et l'utilisation de pipelines bioinformatiques, nous allons nous appuyer sur des exemples issus du projet biomédical européen "understanding the pathophysiology of IntraCranial ANeurysm" (ICAN) dans lequel s'inscrit la thèse.

Ce projet vise à apporter une meilleure compréhension et prédiction du développement et de la rupture des anévrismes intracrâniens, en s'intéressant notamment aux facteurs génétiques et aux mécanismes moléculaires [18, 19]. Dans cet objectif, des biologistes et médecins ont constitué une large base de données d'échantillons biologiques sur une population de 3.000 individus, analysés grâce à des protocoles d'analyse de données ou pipelines. Cette base de données est composée de données de natures diverses, comme des données de séquençage génomiques, des données de neuroimagerie vasculaire cérébrale, mais également des données cliniques (habitudes de vie, antécédents familiaux). La nature de ces données, multiscalaires et hétérogènes, rend leur analyse plus complexe car elle requiert plusieurs expertises spécifiques (à chacun des types de données), mais aussi une grande variété d'outils pour les traiter. En conséquence, l'implémentation, l'exécution et particulièrement le partage des pipelines de traitement des données au sein des équipes (mais également de la communauté) sont des tâches complexes.

2.3 . Contenu des interviews

Les membres du projet ICAN ont des compétences variées. Certains d'entre eux implémentent des pipelines, d'autres font de l'analyse statistique des données, et d'autres établissent et exécutent des protocoles de recueil de données.
Les membres qui implémentent les pipelines peuvent travailler conjointement avec ceux qui obtiennent les données pour les aider à les analyser. L'implémentation de pipelines nécessite d'avoir une compréhension extensive des données, des connaissances approfondies sur les nombreuses méthodes et outils disponibles pour les analyser mais aussi des compétences de développement.
Les développeurs de pipelines peuvent aussi travailler avec des membres qui souhaitent effectuer des traitements statistiques sur les données. Ils veulent pouvoir conduire des analyses répondant à des questions très précises, mais ne savent pas nécessairement traiter les données brutes. Ils ont donc besoin que ces données soient pré-traitées, afin d'en extraire de nouveaux ensemble de données filtrés et structurés sous le format adapté.
Les développeurs de pipelines peuvent également vouloir eux mêmes analyser des données obtenues dans le cadre du projet, et aussi collaborer entre eux pour développer ces pipelines.

Les pipelines (et leurs résultats) produits par les utilisateurs du projet ICAN sont développés dans l'objectif d'être partagés et réutilisés par les autres utilisateurs du projet au sens large.

Dans cette section, nous allons présenter trois développeurs de pipelines du projet ICAN, ainsi qu'un développeur de pipeline externe au projet, en présentant leur contexte de travail et en décrivant des exemples de pipelines implémentés.

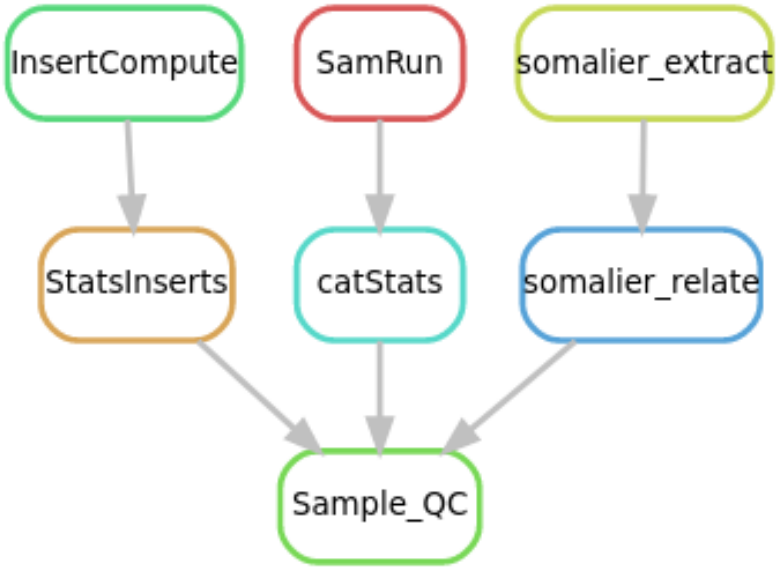

Figure 2.1 – Exemple de pipeline développé par Raphaël, que nous allons présenter plus en détail dans le chapitre suivant.

Raphaël Blanchet, Doctorant

Raphaël Blanchet est un doctorant en bioinformatique de l'unité de recherche de l'ITX, à Nantes.

Contexte de travail. Son projet de thèse s'inscrit dans le projet ICAN. Sa thèse vise à développer une méthode d'analyse pour établir une corrélation génotype-phénotype dans l'apparition des anévrismes intra-crâniens.

Type de données. Dans le cadre de sa thèse, il doit travailler avec des données génomiques, soit des données de séquençage de patients, qui sont sensibles. Les volumes de données traités et manipulés sont importants. Il travaille notamment avec des données de séquençage de génomes entiers, et est amené à générer de gros volumes de données (intermédiaires et finales).

Pipelines produits. Partant de données génétiques brutes, il est nécessaire d'opérer un pré-traitement 2.1 faisant intervenir de nombreux outils de manière efficiente afin d'en permettre l'analyse. Une étude que Raphaël souhaite réaliser est l'étude d'association des variants rares, pour associer des gènes à un phénotype. Pour ce faire, il part de données génétiques de cohortes de patients, qu'il doit dans un premier temps traiter pour en identifier les variants, soit les variations de séquences génétiques. Ce pré-traitement de données contient des étapes de *mapping*, de contrôle qualité comme le marquage de duplicats, et de génotypage (caractérisation

de la variation génétique). De ce "pré-traitement", on obtient un fichier tabulaire contenant l'ensemble des variants génétiques détectés chez chaque individu de la population étudiée. Cette première étape générant bien souvent de nombreux "faux positifs", des contrôles qualité s'imposent donc. Pour ce faire, il a écrit un pipeline qui intègre des étapes de contrôle qualité pour filtrer les variants "faux positifs", des étapes d'annotation de l'impact biologique des variants, et des tests d'association avec le phénotype d'intérêt. Ces pipelines sont longs et complexes, comprennent beaucoup d'outils et d'étapes coûteuses en temps et en ressources (par exemple, des étapes de **mapping**, ou d'alignement de séquences sur un génome, peuvent être très longues).Il doit pouvoir partager facilement ses pipelines avec ses collègues, et pouvoir facilement réutiliser et exécuter les leurs, pour pouvoir comparer et construire ensemble leurs pipelines. Ses résultats doivent être reproductibles, afin d'être validés dans le cadre de leur publication. Il souhaite publier ses pipelines pour partager sa méthode, et que d'autres utilisateurs puissent adapter son pipeline à leurs propres données. Pour ce faire, il met en place des efforts de développement et des scripts pour générer les configurations de ses pipelines pour de nouvelles données.

Pierre Lindenbaum, Analyste de données biologiques

Pierre Lindenbaum est ingénieur à l'unité de recherche de l'ITX, à Nantes.

Contexte de travail. Pierre travaille en collaboration avec d'autres chercheurs dans le projet ICAN. Son travail d'analyse de données consiste à travailler conjointement avec des biologistes pour les aider à analyser leurs données et à implémenter des pipelines.

Type de données. Il analyse des données variées, souvent volumineuses et sensibles, essentiellement de données génomiques de patients.

Pipelines produits. Les analyses que Pierre doit effectuer sont des pipelines de traitement des données de séquençage brutes, et d'interprétations initiales de ces données (par exemple l'annotation fonctionnelle de variants). Il est donc amené à écrire de nombreux pipelines qui peuvent comporter un certain nombre d'étapes plutôt génériques, comme des pipelines dédiées au *mapping*, par exemple. Ce faisant, il se construit une collection de pipelines effectuant toutes les étapes classiques de traitement de données brutes 2.2, ou certaines analyses qu'il conduit régulièrement sur de nouvelles données. Il veut pouvoir réutiliser cette base de pipelines génériques sur plusieurs années. Il peut être amené à réutiliser des portions de code dans de nouveaux pipelines. Il peut aussi être amené à ré-exécuter ces pipelines sur de nouvelles données, et a donc besoin de pouvoir les adapter facilement à de nouveaux cas d'usages. Pierre a aussi besoin de pouvoir maintenir et ré-exécuter facilement ces pipelines dans le temps. Les résultats générés par ses pipelines sont ensuite analysés (dans le cadre d'analyse menées par des statisticiens, par exemple) ou simplement utilisés tels quels par d'autres personnes. Les utilisateurs doivent pouvoir faire confiance à ces résultats, ils doivent être reproductibles. Les utilisateurs doivent également pouvoir comprendre les étapes des pipelines sans documentation extensive.

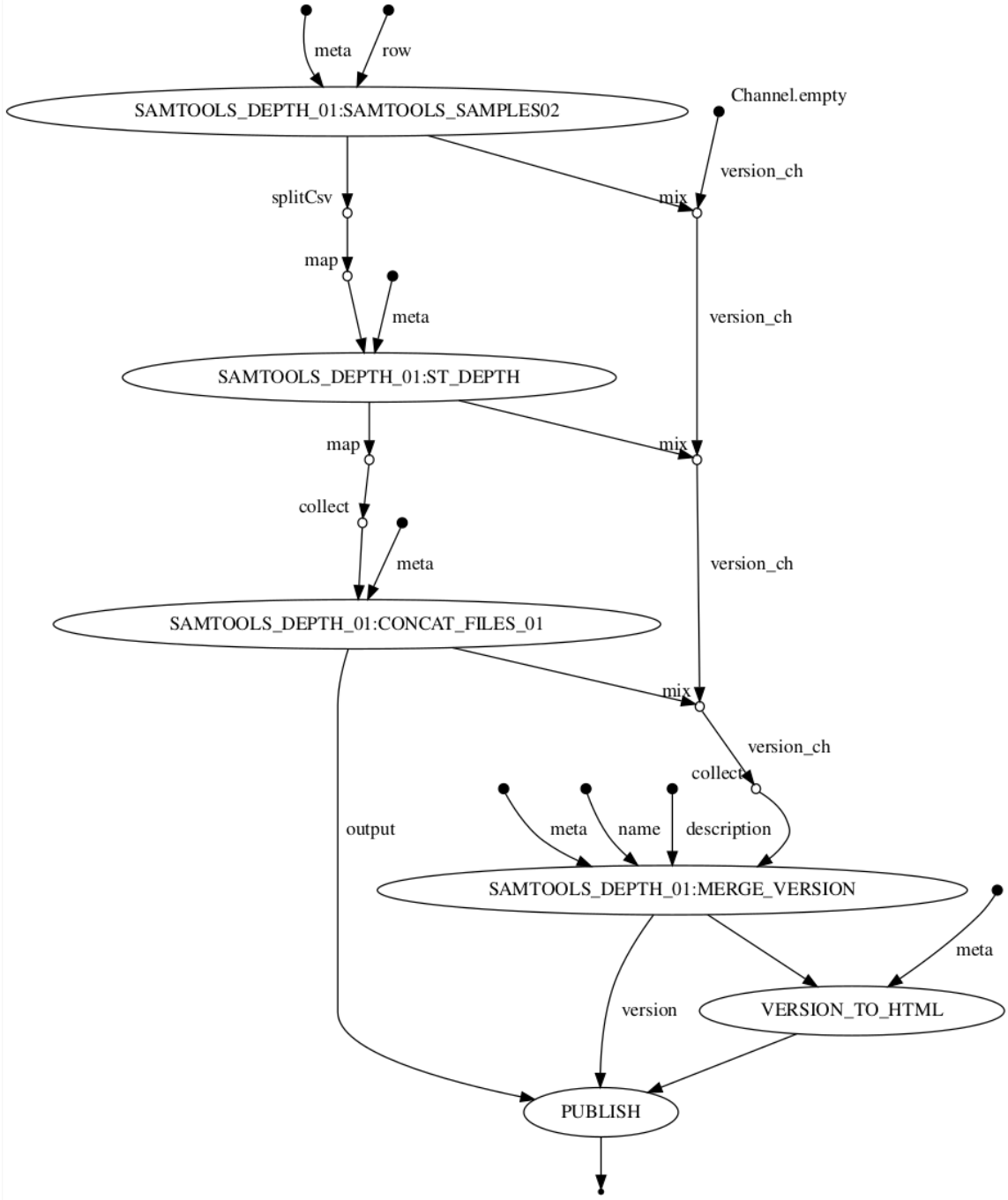

Figure 2.2 – Exemple de pipeline de routine de Pierre qui utilise l'outil Samtools [20]

Éric Charpentier, Formateur

Éric Charpentier est ingénieur à l'unité de recherche de l'ITX, à Nantes.

Contexte de travail Il fait partie de l'équipe de la plateforme BiRD de Nantes, qui propose en collaboration avec la plateforme génomique de Nantes[1] des services de bioinformatique. Cette plateforme met à disposition des pipelines d'analyse de données de séquençage à haut débit, pour standardiser leur analyse, depuis les données brutes jusqu'à l'interprétation biologique. La plateforme propose également une infrastructure de calcul haute performance et de stockage. Cette plateforme aide donc les scientifiques de plusieurs instituts à analyser leurs données non seulement en leur proposant des solutions et une infrastructure mais également en les formant à leur utilisation.

Type de données Les données traitées par ces pipelines sont surtout des données de séquençage, brutes. Les utilisateurs amenés à utiliser ces pipelines ont cependant des données sous des formats similaires, mais de diverses natures.

Pipelines produits Les pipelines produits et mis à disposition par Éric sont surtout des pipelines de traitement des données brutes et d'analyses initiales. Ces pipelines permettent aux utilisateurs de faire des analyses standard. Ces analyses contiennent cependant de nombreuses étapes et sont complexes à implémenter sans les connaissances techniques ou bioinformatiques nécessaires. Éric forme des utilisateurs non développeurs à utiliser ces pipelines. Il doit donc leur inculquer l'ensemble des connaissances nécessaires non seulement pour installer et exécuter les pipelines, mais aussi apprendre aux utilisateurs à les lire et à les comprendre. Ses pipelines doivent donc avoir une documentation extensive. Par ailleurs, une structure standardisée et modulaire sont des caractéristiques importantes pour que des utilisateurs, surtout débutants, puissent s'approprier les pipelines et éventuellement en réutiliser des portions dans d'autres pipelines. Enfin, un effort particulier doit être fait sur la gestion des paramètres du pipeline, pour que ces paramètres puissent être modifiés facilement et ainsi rendre le pipeline transposable à des cas d'utilisation et des données variés. Des paramètres à changer dans les pipelines peuvent être par exemple ceux des outils. Un exemple de pipeline qu'Éric a écrit est un pipeline d'analyse du profilage ARN en séquençage 3' pour l'étude de l'expression génique à partir d'échantillons d'ARN [21]. Il met ce pipeline à disposition dans l'objectif de permettre à d'autres utilisateurs de le réutiliser sur leurs propres données. Ce pipeline traite des données de séquençage brutes (et une description des échantillons), retourne une liste de gènes différentiellement exprimés, et

1. https://pf-genomique.univ-nantes.fr

produit un rapport complet contenant les contrôles qualité effectués et les résultats d'expression. Le pipeline effectue aussi bien des opérations "basiques" de tri et concaténation des fichiers pour les mettre sous des formats adéquats pour les analyses, notamment via des scripts python, que des étapes d'alignement, d'établissement d'une matrice d'expression et de traitement de cette matrice via des outils bioinformatiques dédiés. Le pipeline génère des rapports extensifs contenant les résultats des différentes analyses. Des rapports d'exécution du pipeline, ou données de provenance, garants du bon déroulement de l'exécution, sont également générés pour que les utilisateurs de ces résultats puissent s'y référer. Ce pipeline, qui a été publié, doit être régulièrement mis à jour au gré des changements de version d'outils, ou de changement d'outil de référence pour une opération. Il est donc important pour Éric de pouvoir facilement modifier le pipeline.

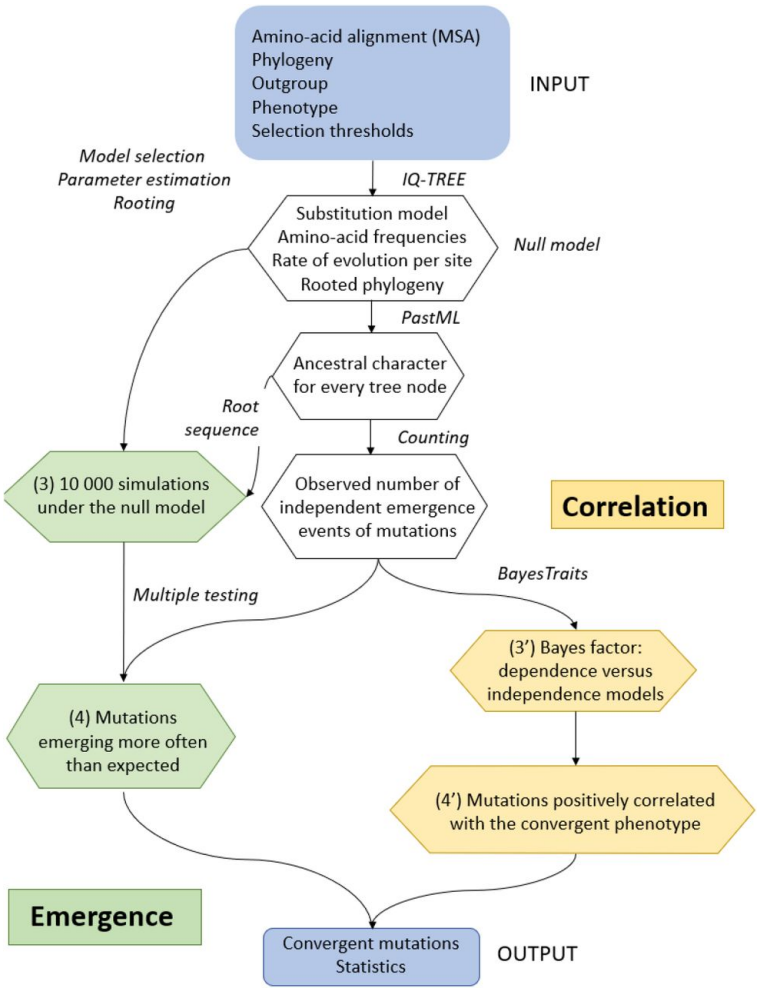

Figure 2.3 – *Flowchart* du pipeline CONDOR de Frédéric Lemoine [22]

Frédéric Lemoine, Data scientist

Frédéric Lemoine est Ingénieur de Recherche et bioinformaticien et travaille au hub bioinformatique et biostatistiques de l'institut Pasteur.

Contexte de travail. Frédéric a deux principales activités : 1) Analyser des données que les biologistes produisent, et 2) développer de nouvelles méthodes d'analyse . Pour cela, il conçoit et implémente des pipelines d'analyse de données, potentiellement conjointement avec les biologistes.

Type de données. Les données que Frédéric traite sont principalement des données issues de séquençage. Il peut traiter des données génomiques de différents organismes, mais ces dernières années principalement des virus. Il peut être amené à travailler avec des données brutes ou des données pré-traitées, mais elles sont généralement très volumineuses.

Pipelines produits. Les analyses de données que Frédéric effectue sont plutôt des pipelines d'analyse spécialisées, comme des analyses statistiques. Dans ses pipelines, on peut retrouver des appels à des outils bioinformatiques, mais surtout des scripts d'analyse de données, notamment en R et en Python. Il co-écrit parfois ces pipelines avec ses collègues, et partage ses pipelines pour qu'ils soient utilisés dans son équipe. Dans ce contexte, avoir un cadre standard d'écriture est important pour faciliter la collaboration sur de tels projets. Un exemple de pipeline écrit par Frédéric est le pipeline CONDOR [22] 2.3. C'est un pipeline d'analyse de données d'évolution, plus précisément de détection des mutations relevant de l'évolution convergente, à partir d'alignements de protéines. Ce pipeline a été établi et testé à partir d'alignements de protéines de poissons passant d'eau douce à eau saumâtre. Ce pipeline fait appel à un certain nombre de scripts, notamment en python, et à très peu d'outils. En effet, c'est un pipeline qui implémente une méthode novatrice, et qui n'effectue pas ou peu d'opérations de routine. Ce pipeline est essentiellement composé d'opérations d'analyse. L'objectif du pipeline est d'être publié en tant que méthode d'analyse. Il faut donc qu'il soit reproductible et facilement réutilisable par d'autres utilisateurs. Il a ainsi été intégré dans un serveur web pour pouvoir être lancé par d'autres utilisateurs sur leurs propres données. Son code source est également disponible sur un dépôt, afin que des utilisateurs puissent l'installer sur leurs propres environnements. Malgré le faible nombre d'outils auquel ce pipeline fait appel, il contient un certain nombre de dépendances logicielles, comme des librairies. Il est donc essentiel de pouvoir rendre cette liste facilement accessible

pour que les utilisateurs soient en mesure d'exécuter le pipeline.

2.4 . Typologie de développeurs et cas d'utilisation

2.4.1 . Typologie de développeurs de pipelines

À partir des pratiques des utilisateurs soulignés dans les interviews, on peut identifier trois axes dans lesquels les utilisateurs se démarquent particulièrement.

Une différence notable entre les différents types d'utilisateurs est le besoin de partager plutôt leurs résultats en tant que tels, ou au contraire leurs pipelines.

Les pipelines produits par les développeurs ainsi que les résultats qu'ils produisent doivent tous être **reproductibles** pour qu'ils soient considérés comme fiables. Comme souligné précédemment, les développeurs ont des pratiques de partage différents. Ils ont donc des besoins différents dans la reproductibilité de leurs pipelines. On peut distinguer trois niveaux de reproductibilité des pipelines inspirés de la littérature [23] :

- refaire : l'utilisateur doit être capable de ré-exécuter son propre pipeline et d'obtenir les mêmes résultats, et des informations détaillées sur l'exécution doivent pouvoir être accessibles ;
- répliquer : un autre utilisateur doit être capable d'exécuter le pipeline, dans un autre environnement logiciel, et d'obtenir des résultats comparables. Le développeur doit donc fournir une documentation détaillée pour faciliter la compréhension du pipeline. Il doit également fournir la liste des dépendances logicielles nécessaires à l'exécution du pipeline ;
- réutiliser : Le pipeline doit pouvoir être utilisé par d'autres utilisateurs sur d'autres environnements logiciels. Il y a deux catégories de "réutilisation", l'adaptation du pipeline pour l'exécuter sur de nouvelles données et la réutilisation de code ou de fragments de code dans d'autres analyses. Pour réutiliser sur de nouvelles données, il est notamment important que l'utilisateur documente son code de manière extensive, et aide à faciliter la paramétrisation.

Plus le niveau de reproductibilité est élevé, plus les efforts à fournir pour l'atteindre de la part du développeur sont grands.

La dernière différence notable entre les utilisateurs est le type d'analyse effectué. On distingue trois grandes catégories d'analyses : le pré-traitement, les analyses initiales et enfin les analyses spécifiques :

- Le pré-traitement correspond à l'étape où les données brutes sont traitées et transformées, et examinées pour produire de premières observations. Les traitements effectués peuvent être de l'ordre du tri, du filtrage ou du contrôle qualité, et des opérations courantes qui sont l'alignement de séquences et la détection de variants génétiques ;
- Les analyses initiales correspondent aux analyses plus avancées effectuées sur ces données traitées. Cela peut inclure des analyses telles que l'assemblage de novo des génomes, l'annotation des gènes, l'identification des régions régulatrices, l'analyse de la structure et de la fonction des protéines, l'analyse des voies métaboliques, *etc* ;
- Les analyses spécifiques correspondent aux étapes d'intégration et d'interprétation des résultats obtenus lors du pré-traitement et des analyses initiales dans un contexte biologique plus large. Cela peut impliquer la comparaison des données obtenues avec des ensembles de données publiques, l'identification de motifs ou de signatures spécifiques, l'analyse des réseaux génétiques ou protéiques, l'exploration de données à grande échelle, *etc*, dans le but d'apporter des *insights* pertinents.

Les pipelines de pré-traitement et d'analyses initiales comportent beaucoup d'étapes algorithmiquement complexes et "standard" (alignement, contrôle qualité,...) et font donc appel à plus d'outils que les pipelines d'analyse spécifiques. Ces derniers effectuent des opérations moins standards et font donc appel à plus de scripts *ad-hoc*.

D'après les différents besoins des développeurs sur ces axes, on peut identifier une typologie d'utilisateurs, résumée dans le tableau 2.1.

L'ingénieur plateforme écrit et exécute des pipelines pour répondre aux besoins d'analyses d'autres personnes qui ne savent pas les implémenter. Ces pipelines vont effectuer des tâches de pré-traitement et d'analyse initiale sur des données brutes. Ces pipelines comportent beaucoup d'étapes de traitement standard des données et font donc appel à beaucoup d'outils, et il est important pour cet utilisateur de trouver les outils adaptés. L'ingénieur plateforme a besoin que les résultats produits par ses pipelines soient fiables et donc reproductibles, et de pouvoir reconduire ses propres analyses. Il a également besoin de garder une trace de l'exécution des pipelines, notamment pour que les futurs utilisateurs de ces données puissent comprendre comment elles ont été générées. Comme beaucoup des pipelines de l'ingénieur plateforme comportent des étapes de pré-traitement standards, il peut avoir besoin d'en réutiliser certains de façon récurrente.

L'analyste de données écrit des pipelines pour analyser ses propres données.

Il souhaite conduire des analyses approfondies, et doit donc écrire des pipelines comportant beaucoup de scripts sur mesure. Ses pipelines ont en général moins d'outils, car il effectue moins d'opérations standards et qu'il part de données pré-traitées. Son objectif est de rendre disponible sa méthode. Il veut permettre à d'autres développeurs de réutiliser ses pipelines, soit en les adaptant à de nouvelles données, soit en les réutilisant dans de nouveaux pipelines. Il rend ses pipelines disponibles sur des dépôts sur lesquels ils peuvent être identifiés, afin que d'autres puissent les trouver et les citer. Ces pipelines devront être maintenus pour qu'ils puissent être utilisés sur le long terme.

Le formateur souhaite écrire des pipelines qui puissent être réutilisées par un grand nombre d'utilisateurs n'ayant pas forcément de compétences de développement extensives. Les pipelines doivent donc avoir une structure standardisée et facile à appréhender, et surtout modulaire pour qu'ils soient facilement modifiables et adaptables à d'autres contextes. Il est important que les pipelines produisent des résultats reproductibles. Il est également important que l'installation des pipelines soit la plus simple possible. De façon générale, le formateur veut pouvoir toucher un public provenant de communautés plus étendues, moins familiers avec le développement de pipelines.

profil	ingénieur plateforme	analyste de données	formateur
partage	résultats	pipeline	pipeline
niveau de reproductibilité (1)	*/**	**/***	***
type d'analyse (2)	I/II	III	I/II

Table 2.1 – Tableau présentant des axes de comparaison des différents types d'utilisateurs
(1) * : refaire, ** : répliquer, *** : réutiliser
(2) I : pré-traitement, II : analyses initiales, III : analyses spécifiques

2.4.2 . Cas d'utilisation : Besoins utilisateurs

Nous allons ici illustrer les besoins des utilisateurs de pipelines en nous appuyant sur des exemples tirés des utilisateurs précédemment décrits. Nous présentons les cas d'utilisations de façon centrée sur un profil d'utilisateur et mettons en évidence les autres profils potentiellement confrontés au même cas d'utilisation.

Nous commençons ici par deux cas d'utilisation centrés sur le profil le plus générique : ingénieur plateforme (incarné par Pierre et Raphaël).

Conception de pipelines (CU1) Lors de la conception du pipeline, Raphaël a besoin de savoir quels outils choisir pour traiter ses données de séquençage. Il utilise le guide de référence afin de savoir quels outils sont considérés comme de référence pour les traitements de données qu'il veut effectuer. Le problème du choix du bon outil pour effectuer une opération donnée est un problème partagé par la communauté des bioinformaticiens. En sachant quels sont les traitements de données qu'ils effectuent, par exemple, ils veulent pouvoir trouver des outils répondant à leurs besoins et être en mesure de les comparer entre eux.

Maintenance (CU2) Pierre utilise le même pipeline d'analyse de données depuis des années. Son pipeline comporte beaucoup d'outils, et ces outils peuvent parfois changer de version ou devenir obsolète. Il souhaite maintenir facilement son pipeline et changer de façon simple les outils individuels si besoin, dans le cas de changement de version par exemple. De façon générale, les développeurs ont besoin de rendre leurs pipelines capables d'être ré-exécutés dans le temps, par eux-mêmes ou par d'autres, afin qu'il soit reproductible et de pouvoir faire confiance au pipeline ou aux résultats produits par celui-ci.

Complétion de pipelines (CU3) Pierre trouve de nouveaux outils à intégrer dans ses pipelines lorsqu'ils sont publiés. Il veut en effet utiliser des outils très récents, notamment pour être à jour de l'état de l'art. Cependant, l'intégration de ces outils à des pipelines n'est pas toujours simple, la documentation n'étant pas toujours extensive par exemple. Il cherche donc dans les pipelines disponibles sur des dépôts des exemples d'implémentation de ces outils afin de s'en inspirer et d'adapter ces exemples à son cas. Lors de l'écriture de pipelines les développeurs peuvent avoir besoin de compléter leurs pipelines avec des portions d'autres pipelines.

Dans la suite, nous introduisons deux cas d'utilisation très présents dans les profils formateurs (représentés par Éric).

Provenance des données (CU4) Il arrive que des analystes de données utilisant les pipelines d'Éric lui demandent des années plus tard des précisions sur l'exécution ou la version de leur pipeline. Cependant, ses pipelines génèrent un rapport détaillé d'exécution, et le code source des pipelines est à disposition des utilisateurs. Un moteur d'exécution comportant un module de génération d'informations du déroulement d'exécution répond donc à un besoin d'analystes de

données. Les analystes de données travaillant avec des pipelines produits par des tiers ne sont pas forcément en capacité de les développer relire ou modifier eux mêmes. Ils peuvent avoir besoin d'informations sur le déroulement de l'exécution pour faire confiance aux résultats.

Adaptation des pipelines aux données (CU5) Éric cherche à rendre certains de ses pipelines d'analyses routinières disponibles pour d'autres utilisateurs, afin qu'ils puissent les exécuter sur leurs propres données. Cependant, pour pouvoir les adapter à leurs propres données, il leur faut pouvoir changer les paramètres de certains outils, en changeant la configuration du pipeline, notamment via les paramètres des outils. Pour ce faire, ils auraient besoin d'aide à la recommandation de paramètres, qu'Éric réunit dans un fichier dédié lu par le pipeline et pour lequel il produit une documentation extensive, mais ce travail demande des efforts conséquents. Les utilisateurs, notamment les analystes de données, peuvent vouloir réutiliser un pipeline sur leurs propres données, mais tous les pipelines n'ont pas de fichiers de paramètres dédiés, et l'adaptation de la plupart des pipelines à de nouvelles données nécessite une compréhension des outils utilisés et de leur implémentation dans le script du pipeline.

Enfin, les cas d'utilisation ci-dessous sont associés aux analystes de données (représentés par Frédéric).

Test de pipelines (CU6) Frédéric souhaite partager son pipeline CONDOR pour permettre à d'autres utilisateurs de l'exécuter sur leurs propres données. Il doit donc s'assurer de la fiabilité de son pipeline, et garantir que son pipeline a un comportement attendu. Il a donc besoin de tester ses pipelines, et de constituer des jeux de données test représentatifs.

Calcul haute performance (CU7) Frédéric a écrit le pipeline CONDOR qu'il a exécuté sur des données "test" de volume modeste, mais qui est destiné à fonctionner sur des volumes de données potentiellement très larges. Son pipeline contient des étapes qui peuvent être coûteuses en ressources de calcul et en temps. Son pipeline doit donc pouvoir passer à l'échelle, et pouvoir tourner sur des environnements de calcul haute performance. De façon générale, les bioinformaticiens sont amenés à travailler sur des volumes de données importants, et beaucoup d'étapes peuvent être coûteuses en ressources (comme les étapes de mapping, par exemple). Ils ont besoin de pouvoir implémenter leurs pipelines sur

des environnements adaptés, en optimisant les calculs et tâches nécessaires.

Établissement d'un état de l'art (CU8) Frédéric produit des pipelines d'analyse très spécifiques et très peu routiniers, pour lesquels les méthodes d'analyses sont très variées. Il veut pouvoir se positionner par rapport à l'existant. Cependant, les pipelines ne sont pas tous publiés dans la littérature, mais leur code peut être disponible sur des dépôts. Il veut donc explorer ces dépôts et retrouver des pipelines grâces aux opérations qu'elles effectuent. Les développeurs de pipelines, mais également les analystes de données, peuvent avoir besoin d'établir un état de l'art afin de prendre connaissance des pratiques et méthodes d'analyses en cours pour effectuer certaines analyses. Cette liste pourrait être ordonnée ou partitionnée par type d'outils, par exemple, pour faciliter le travail de tri.

Citation de pipelines (CU9) Frédéric, en plus de rendre le pipeline CONDOR disponible en tant que service web, publie son code sur GitHub. Son pipeline est également publié dans la littérature, donc il est simple pour un utilisateur tiers de citer le pipeline. Les développeurs de pipelines peuvent vouloir partager leur code pour qu'il puisse être réutilisé par d'autres, mais veulent pouvoir être cités en cas de réutilisation de leur code. Cependant, dans le cas d'un pipeline non publié dans un journal et uniquement disponible sur un dépôt, et ayant plusieurs versions stables, il est plus compliqué de citer le pipeline.

2.5 . Cycle de vie d'un pipeline

Les interviews des utilisateurs nous ont permis de lister leurs pratiques de développement, d'exécution et d'utilisation des pipelines bioinformatiques, à partir desquelles nous avons défini des étapes de leur "cycle de vie". Ce cycle de vie est introduit dans l'article [17]. Nous allons ensuite décrire les difficultés associées à chacune de ces étapes à partir d'exemples du projet R2-P2 2.2. Pour définir ces difficultés, nous nous sommes basés sur les cas d'utilisation du chapitre2, que nous avons associés à chacune de ces étapes.

Le cycle de vie des pipelines, tel qu'illustré dans la figure 2.4 contient de nombreuses étapes qui présentent chacune nombre de challenges pour les développeurs, mais également pour les analystes de données.
Même si tous les pipelines ne sont pas voués à être partagés et utilisés par le plus grand nombre, il est nécessaire de s'assurer qu'ils produisent des résultats fiables et *reproductibles* et qu'ils utilisent les outils et scripts appropriés avec les paramètres

Figure 2.4 – Cycle de vie d'un pipeline bioinformatique [17]. Les cas d'utilisation sont disposés à côté des étapes associées. Les cas d'utilisation roses, oranges et jaunes sont respectivement ceux des profils de l'ingénieur plateforme, du formateur et de l'analyste de données.

adaptés, pour être en mesure de faire confiance aux résultats produits.

2.5.1 . Développement des pipelines

La première catégorie de problèmes rencontrés est liée au développement concret du pipeline, qui implique généralement l'utilisation de plusieurs langages de programmation et environnements logiciels différents. Développer un pipeline peut être ardu, les analyses de données bioinformatiques comportant typiquement plusieurs étapes complexes et de natures différentes (scripts de divers langages, grande variété d'outils). Ces étapes peuvent être des étapes de **traitement** des données et ou bien d'**analyse** de données.
On peut distinguer deux catégories de difficultés dans le développement de pipelines. La première catégorie de difficultés est la complexité des environnements de programmation, liée à la variété de langages intervenant dans les pipelines, mais également la variété des outils.

Dans le projet ICAN par exemple, des données de séquençage sont traitées en utilisant une multitude d'outils (*e.g.*, BWA [24], GATK [25], Picard Tools [26]), et les données ainsi traitées sont utilisées pour des analyses statistiques effectuées par le biais de Python et de librairies R dans des Jupyter Notebooks[2] [27].
La deuxième catégorie de difficultés est la difficulté du choix des outils.
En effet, il y a une abondance d'outils bioinformatiques étant publiés, parfois plusieurs effectuant des tâches similaires. Ainsi, au delà du choix des traitements à effectuer au sein de chacune des étapes, le choix du bon outil complexifie le développement des pipelines.

2.5.2 . Test des pipelines

Le test des pipelines, comme le test de code en général, est une étape longue mais nécessaire pour contrôler la fiabilité d'un pipeline [28]. Dans un premier temps, il est important de s'assurer que le pipeline fonctionne correctement dans son environnement d'exécution personnel. On veut pouvoir s'assurer que chacune des étapes fonctionne séparément, mais également que l'enchaînement des étapes (sous parties du pipeline et pipeline entier) fonctionne correctement et produit les résultats escomptés. Le choix de cas test, soit la sélection de jeux de données test à donner en entrée est également un problème de taille pour les développeurs.
Il est possible de constituer un jeu de données test en sélectionnant un sous ensemble du jeu de données initial. Cela requiert des efforts et de l'expertise de la part du développeur de pipelines. Par exemple, dans le cas d'un jeu de données de séquençage de génomes, il peut être difficile de sélectionner les échantillons qui soient assez représentatifs (pour pouvoir tester les paramètres ou le bon fonctionnement des outils) mais de taille assez réduite (pour pouvoir diminuer significativement le temps d'exécution du pipeline).
Dans un second temps, s'assurer que le pipeline puisse fonctionner dans différents environnements et sur différentes plateformes, et qu'il puisse être exécuté sur des jeux de données de tailles différentes, est difficile de par la complexité des environnements logiciels nécessaires pour l'exécuter.
Dans le **CU6**, Frédéric doit tester ses pipelines pour s'assurer de leur fiabilité.

2.5.3 . Déploiement

Les analyses de données bioinformatiques sont souvent dédiées à l'analyse de jeux de données de grande taille, et effectuent des tâches coûteuses. Elles ne peuvent ainsi pas tourner en local ou sur des serveurs simples mais nécessiteront des infra-

2. https://github.com/ICAN-aneurysms/RIA-predict

structures de calcul de haute performance (HPC). Le déploiement et l'adaptation des pipelines sur ce type d'environnement peuvent être des étapes très longues, ardues et nécessitant une expertise technique particulière [29]). Il est en effet nécessaire de :

- surveiller la soumission de tâches (par exemple, en exécutant les commandes correctes, avec le bon nombre de ressources comme la mémoire ou le CPU nécessaire) ;
- s'assurer que les travaux se sont exécutés correctement, et les soumettre à nouveau le cas échéant.

2.5.4 . Maintenance

Maintenir la cohérence et la robustesse du pipeline au fil du temps est crucial. Cependant, compte tenu de la complexité des pipelines, un pipeline qui fonctionnait auparavant peut subir une panne ou produire des résultats inattendus en raison de l'une ou plusieurs des causes suivantes :

- des changements dans l'environnement matériel ;
- des changements dans l'environnement logiciel (*e.g.*les outils ne sont plus disponibles ou ont été mis à jour) ;
- des changements dans les ensembles de données (par exemple, les séquences de génomes de référence évoluent constamment).

Cela peut nécessiter des mises à jour régulières du pipeline lui-même.

2.5.5 . Reproductibilité

Pour qu'un pipeline soit reproductible, il est nécessaire (mais pas suffisant) de pouvoir l'exécuter dans le temps (maintenance) et sur plusieurs sites différents (déploiement). Un pipeline qui fonctionne avec succès sur un site peut ne pas fonctionner sur un autre site, ou peut produire des résultats inattendus en raison d'un planificateur de *cluster* différent, de matériel ou d'un environnement logiciel différent. De façon complémentaire, il est nécessaire de contrôler et rendre disponibles trois composantes pour s'assurer de la reproductibilité : le code des pipelines, les données mais aussi l'environnement logiciel. Ce dernier paramètre est le plus complexe à contrôler.

Besoin de reproductibilité dans le contexte d'ICAN. Dans le projet ICAN, les données analysées sont des données d'analyse médicales, qui sont soumises à des réglementations de protection des données personnelles. Ainsi, ces données ne sont

pas simples, voir impossibles, à partager. Or, les protocoles d'acquisition des données peuvent eux être partagés par les différentes parties du projet. De cette façon, pour permettre le partage et les échanges il faut s'assurer que les pipelines soient déployables chez chacun des différents acteurs. Il est donc extrêmement important de s'assurer que les pipelines (i) s'exécutent correctement et (ii) retournent les résultats escomptés sur sur différentes plateformes HPC (qui seraient différentes de leurs environnements de développement et de test). Dans ce contexte, l'utilisation de standards pour la spécification et configuration des pipelines ainsi que l'utilisation de langages et outils open source sont essentielles pour s'assurer de la reproductibilité.

La reproductibilité est donc un pré-requis majeur pour s'assurer de la fiabilité des pipelines mais aussi des résultats qu'ils produisent. Par ailleurs, pour attester de la fiabilité de résultats obtenus, il est nécessaire de garder une trace de l'exécution. Des métadonnées ou des rapports d'exécution peuvent répondre à ce besoin.

2.5.6 . Réutilisation

Un pipeline ne peut être réutilisé que s'il est reproductible. En d'autres termes, la reproductibilité est la base de la science cumulative : si le pipeline a été conçu pour être reproductible, il y a de l'espoir qu'il soit plus facilement partagé et réutilisé (en partie ou en totalité) par des tiers.
Six besoins doivent être satisfaits pour la réutilisation.

Tout d'abord, pour que les pipelines soient partagés et réutilisés, ils doivent être facilement compréhensibles pour déterminer leur pertinence et la façon de les réemployer dans d'autres analyses. La documentation du pipeline et sa modularité (où les pipelines ne sont pas écrits comme du code/script linéaire) jouent ici un rôle clé. Si les tâches sont encapsulées dans des éléments constitutifs "indépendants" du pipeline, de façon similaire à une fonction, il est plus simple de les isoler pour les adapter à d'autres contextes.

Deuxièmement, pour que des utilisateurs puissent ré-exécuter un pipeline sur de nouvelles données, il faut les aider à identifier les modifications à apporter au pipeline et à ses paramètres. Lorsqu'un utilisateur souhaite réutiliser un pipeline pour de nouveaux jeux de données, il lui faut comprendre de manière approfondie le code du pipeline. Le choix des paramètres des nombreux outils intervenant dans le pipeline peuvent également poser problème aux utilisateurs.

Troisièmement, une fois développés, les pipelines devraient être mis à disposition dans des registres qui permettent aux développeurs de partager avec une équipe ou une communauté et de documenter des pipelines. Cette mise à disposition des pipelines peut aider les utilisateurs à retrouver plus facilement des pipelines qui correspondent à leurs besoins.

Quatrièmement, les registres de pipelines doivent être en mesure de suivre l'historique d'un pipeline au fil du temps. En effet plusieurs versions et modifications d'un pipeline peuvent être développées pour s'adapter aux changements dans l'environnement logiciel, dans l'environnement matériel et dans les besoins fonctionnels.

Cinquièmement, en plus de partager les pipelines, le partage de leurs modules constitutifs, ou *processeurs*, peut aider les développeurs à gagner du temps précieux pendant le développement. Ces processeurs devraient être documentés avec des métadonnées et des annotations afin d'être facilement trouvés.
Enfin, une incitation pour que les développeurs partagent leur pipelines est de les rendre "citables", notamment dans le cas où le pipeline est uniquement disponible sur un dépôt.

2.6 . Conclusion

Dans ce chapitre, nous avons présenté les pipelines bioinformatiques au travers du cas d'utilisation du projet R2-P2 et décrit et classifié des profils de développeurs de pipelines de ce cas d'utilisation.
Les pipelines, qui sont destinés à analyser des données souvent hétérogènes et volumineuses, sont des objets complexes à concevoir et à implémenter. Les développeurs de pipelines doivent maîtriser plusieurs connaissances et compétences de développement mais aussi d'analyse de données bioinformatique pour les écrire.
Les pipelines doivent répondre à des besoins variés des utilisateurs, en particulier de pouvoir être reproductibles (produire des résultats de confiance) et réutilisables (pouvoir être ré-adaptables dans de nouveaux contextes et sur de nouvelles données).

3 - État de l'art

Sommaire

3.1 . Introduction

Dans ce chapitre, nous introduisons et explorons les solutions disponibles dans la littérature aux problèmes recensés dans le chapitre 2.

Dans un premier temps (3.1), nous présenterons les solutions disponibles pour les bioinformaticiens pour l'écriture de leurs pipelines.

Dans un second temps (3.2), nous introduisons des éléments de solutions génériques pour chaque étape du cycle de vie des pipelines (figure 2.4).

Nous nous concentrerons ensuite sur les solutions à ces problèmes proposés par les systèmes de workflows scientifiques (3.3).

En fin, nous concluons (3.4).

3.2 . Implémentation de pipelines

Dans cette section, nous allons présenter les principales manières d'implémenter des pipelines en bioinformatique. Nous allons commencer par parler des prédécesseurs des *frameworks* dédiés à la construction de pipelines : les scripts.

3.2.1 . Scripts

Les bioinformaticiens peuvent implémenter leurs pipelines sous forme de scripts, pour orchestrer leurs différents outils et analyses.

Ces scripts peuvent être écrits dans des langages comme Perl ou bash. Le scripting permet de construire des pipelines flexibles, suivant une approche impérative.
C'est une manière simple d'écrire un pipeline, mais qui possède de nombreuses limites. À mesure que les pipelines deviennent plus complexes, la gestion des dépendances logicielles peut devenir difficile et entraîner des problèmes de compatibilité. Les scripts peuvent ne pas être facilement portables entre différentes plateformes ou systèmes d'exploitation, ce qui peut rendre difficile leur exécution sur différentes infrastructures. De même, les scripts spécifiques à un projet particulier peuvent avoir une structure qui ne les rend pas adaptables à des projets différents : il n'existe pas de cadre de standardisation d'écriture de pipeline pour les scripts. Enfin, optimiser les scripts pour une exécution à grande échelle n'est pas simple, ce qui peut limiter leur efficacité pour le traitement de gros volumes de données.

Deux fonctionnalités clés nécessaires au traitement efficace des données manquent aux pipelines sous formes de scripts : la prise en charge des "dépendances" (entre les tâches ou les fichiers) et la reprise sur erreur (capacité d'un programme à reprendre là où il s'est arrêté en cas d'interruption). Les pipelines incluent souvent des étapes qui échouent pour diverses raisons tels que des problèmes de réseau ou de disque, une corruption de fichier ou des bugs. Un pipeline doit être capable de récupérer à partir du point de contrôle le plus proche plutôt que de supprimer ou d'écraser des fichiers intermédiaires par ailleurs utilisables. De plus, l'introduction de nouveaux fichiers en amont dans une analyse, tels que des échantillons, ne doit

pas nécessiter la reprise du traitement des échantillons existants.

Note sur l'utilitaire Make L'utilitaire Make, créé en 1977 par Bell labs, permet d'automatiser la compilation et la construction de logiciels ou de projets, selon une logique déclarative. C'est un système très efficace pour définir et maintenir des dépendances simples et donc pour automatiser des tâches récurrentes, et a été adopté comme solution populaire pour implémenter des workflows lors de l'explosion des outils bioinformatiques [30]. Il permet également la reprise sur erreur. Cependant, Make n'est pas doté de fonctionnalités suffisantes : il ne favorise pas la modularité des pipelines, leur portabilité ni leur reproductibilité, et ne permet de gérer que des dépendances simples.

3.2.2 . Notebooks

Un notebook est un document interactif qui permet de mélanger du texte, des équations mathématiques, des graphiques, des médias interactifs et du code exécutable dans un environnement de programmation.
Dans un notebook, le code peut être exécuté directement dans le document, ce qui permet de visualiser immédiatement les résultats et d'explorer les données de manière interactive. Cette transparence peut permettre aux utilisateurs de mieux comprendre comment des résultats sont obtenus, surtout dans le cadre d'analyses statistiques, par exemple.

Les notebooks peuvent être partagés facilement, car ils peuvent être sauvegardés sous forme de fichiers texte, .md,... et ils peuvent être publiés en ligne. Les notebooks sont ainsi de plus en plus utilisés comme moyen de partager et d'afficher du code source tout en fournissant une visualisation interactive des résultats [27].

Le langage de notebooks pionnier est Mathematica [31], développé en 1988 par Wolfram research. C'est le premier notebook qui a permis de mélanger du texte, du code, des images/médias et des formules mathématiques. Il permet notamment d'exécuter du code du langage propriétaire de Wolfram research, même si d'autres langages sont compatibles avec Mathematica.

Pour le langage R, un autre langage couramment utilisé en bioinformatique, plusieurs packages ont été développés permettant d'écrire des notebooks.

Le premier *package* R permettant d'écrire des Notebooks est Sweave [32], développé en 2002 par F. Leisch pour le R project. Ce *package* permet de combiner du R et du Latex, pour générer des rapports en Latex en exécutant le code R de manière dynamique. Les utilisateurs de R étant en général aussi utilisateurs de Latex, c'est une façon abordable pour la communauté de générer ces rapports sans trop de difficulté pour les utilisateurs.

Une solution plus récente, le *package* knitr [33], créé en 2012 par Yihui Xie est le successeur de Sweave. Ce *package* permet lui aussi de générer des documents (des rapports, notamment en pdf ou html), en intégrant du code R (mais aussi d'autres langages, comme Julia, python,...) et du texte. Il va compiler le code et le "tricoter" avec le texte pour générer un rapport. Il est possible de spécifier un grand nombre d'options dans les balises pour personnaliser l'affichage du code et des résultats. Il est également plus performant que Sweave et permet d'implémenter des pipelines plus polyvalents.

R Markdown [34], créé en 2013 par le R project, est une extension de Markdown. C'est un langage de balisage plus léger que knitr. Il permet lui aussi d'intégrer du code R exécutable dans du markdown, et permet également l'utilisation d'autre langages comme python, julia, c++, sql,... Il propose une syntaxe intuitive pour la création de notebooks. Les fichiers rmarkdown peuvent être "compilés" avec knitr pour générer plusieurs formats de document (comme cité précédemment, html, pdf,...).

La solution la plus connue et qui a popularisé le concept de notebooks est Jupyter Notebook [35], lancé par le projet Ipython en 2014. C'est un format de document ouvert basé sur json pour faire des pipelines reproductibles, qui se voulait à l'origine être une version libre des notebooks Mathematica [36].

Ces notebooks fournissent un environnement de développement interactif, qui aide à visualiser les résultats lors de l'exécution du code. Le code (ou le texte) est écrit dans des cellules (de type "code" ou "texte"). Les cellules texte sont au format Markdown, et les cellules de code peuvent exécuter du code de plusieurs langages, le plus populaire étant python. Il n'est possible d'utiliser qu'un langage de programmation à la fois dans les cellules "code" d'un notebook, même s'il est toujours possible de lancer du code shell dans des cellules "terminal". Ainsi, ces notebooks sont très polyvalents, permettant de lancer des outils en ligne de commande mais également des scripts directement sur l'interface. Les widgets Jupyter

sont des éléments supplémentaires pour interagir avec les notebooks. Ce sont des contrôles de formulaire comme des curseurs, des champs de texte, mais aussi des fonctionnalités de visualisation en 2D ou en 3D.

Les cellules des notebooks peuvent être lancées indépendamment. Ce design peut aider lors de la phase de développement de pipelines, permettant aux utilisateurs de tester plus facilement des fragments de code, sans avoir besoin de compiler la totalité du notebook [37]. Cet avantage peut devenir un inconvénient, car il est aussi plus simple d'écrire le notebook dans un ordre non linéaire par inadvertance lors d'une phase de test, ce qui nuit à la reproductibilité du notebook [38].

3.2.3 . Workflows

Dans la suite de cette section nous décrivons tout d'abord plusieurs systèmes de workflows populaires dans tous les domaines de la bioinformatique. Dans un second temps, nous décrirons d'autres systèmes de gestion de workflows notables, concernant non seulement la communauté bioinformatique mais également d'autres communautés plus éloignées (finance,*etc*). Cette section est inspirée des descriptions de systèmes de workflows de l'article [11].

Les systèmes de gestion de workflows en bioinformatique

Taverna Taverna [39], créé en 2004 par le projet myGrid et arrêté en 2020, est le système de workflows pionnier en bioinformatique. Taverna a également été adopté par les communautés de Biodiversité, de Sciences sociales et d'Astronomie [5].

Spécification du workflow
Dans un contexte de multiplication de web services (c'est-à-dire d'outils) bioinformatiques, vers lesquels les bioinformaticiens devaient se tourner pour effectuer leurs analyses, il devenait fastidieux de devoir opérer "à la main" un grand nombre de web services. Taverna vise à faciliter l'enchaînement de ces tâches en permettant en prime aux utilisateurs de créer des workflows reproductibles.
Le langage dans lequel les workflows Taverna sont écrits est Scufl. Il s'agit d'un langage conceptuel basé sur XML [39], dans lequel chaque étape du workflow représente une tâche ou un processeur.

Cependant, les utilisateurs de Taverna n'ont pas besoin de coder leurs workflows en Scufl. Taverna possède un "workbench" [16], une interface graphique qui leur permet d'agencer les processeurs, sélectionnés depuis une liste de "services", en les reliant via leurs inputs en outputs. Par défaut, Taverna fournit une liste de services, mais les utilisateurs peuvent en ajouter eux mêmes. Les processeurs peuvent être de plusieurs types selon les opérations effectuées, et peuvent même être des sous workflows. Les processeurs peuvent aussi être associés à des scripts locaux.

Contexte et environnement d'exécution
Taverna peut se lancer en ligne de commande, ou via le Taverna workbench.
Il est possible d'exécuter les workflows Taverna sur des infrastructures de calcul haute performance grâce à des plugins. Cependant, l'optimisation de l'exécution des tâches n'est pas facilitée par ce système de gestion de workflows.

Contexte et environnement d'exécution
Bien que Taverna soit axé sur la reproductibilité, il ne possède pas de mécanismes de capture des dépendances (comme au travers de visualisation avec Docker, par exemple).

Galaxy Galaxy [13], lancé en 2005 par le Galaxy project, est un autre système de gestion de workflows encore très populaire en bioinformatique.

Spécification du workflow.
Galaxy fonctionne en tant que webserveur. L'interface utilisateur est accessible depuis un navigateur, et la composition de workflows se fait depuis cette interface. Comme Taverna, Galaxy propose une interface graphique de composition de workflows, où les processeurs utilisent des outils provenant du Toolshed. Le Toolshed est une bibliothèque centralisée où des utilisateurs "développeurs" peuvent implémenter de nouveaux outils, qui pourront être utilisés pour constituer des pipelines par tous les utilisateurs d'un même serveur.

Cependant, Galaxy a été développé dans un contexte où les créateurs de workflows n'avaient pas nécessairement des compétences de développeurs. Ainsi, même si les utilisateurs ne savent pas coder, ils peuvent aisément composer leurs workflows et régler les paramètres de leurs outils. Pour ajouter des outils au Toolshed, il faut qu'ils soient créés par des développeurs. Il existe plusieurs instances publiques de Galaxy sur lesquelles les utilisateurs peuvent créer et faire tourner leurs workflows.

Les processeurs des workflows peuvent également être des sous workflows.

Ainsi, même si Galaxy est en principe très personnalisable, il est largement utilisé en bioinformatique, les outils des instances publiques de Galaxy étant orientées sur ce domaine.

Une fonctionnalité intéressante de Galaxy est la capacité de parcourir directement les bases de données bioinformatiques publiques, en plus de celles que l'utilisateur peut ajouter.

Exécution du workflow
Les instances Galaxy comprennent typiquement le logiciel Galaxy, une base de données, des outils d'analyse, des workflows personnalisés, et une interface utilisateur. Bien qu'il soit possible d'installer Galaxy sur divers environnements, de l'environnement individuel aux serveurs partagés, ces instances sont typiquement dédiées à être partagées par un grand nombre d'utilisateur. En effet, avoir une instance Galaxy sur un serveur partagé leur permettra de mutualiser le Toolshed et diminuera les coûts d'échelle de maintenance et de mise en place du serveur.
Malgré ces avantages, et sa grande facilité d'appréhension par les auteurs des workflows, Galaxy présente également certaines limitations, notamment en termes de complexité technique, de standardisation, de réutilisation du code et de partage des résultats [11].

Contexte et environnement d'exécution.
Bien que l'interface graphique utilisateur (GUI) de Galaxy offre un puissant support pour l'implémentation de pipelines de novo par des non-spécialistes, elle impose également une lourde charge de développement car, même si il est possible d'importer un workflow Galaxy d'une autre instance, les outils de ce workflow devront être réimplémentés dans le Toolshed de la nouvelle instance. Cela peut être très exigeant dans le cas de pipelines complexes. Les fichiers de spécification des workflows Galaxy ne contiennent en effet pas toutes les informations nécessaires pour les exécuter sur une autre instance [11].

Snakemake. Snakemake [14], créé en 2012 par Johannes Koster et Sven Rahman, est un système de workflows très populaire aujourd'hui en bioinformatique.

Spécification du workflow.
Les workflows Snakemake sont écrits dans un Domain Specific Language (DSL). Snakemake est écrit en python et son DSL a une syntaxe proche de ce langage, et permet d'utiliser la syntaxe et les fonctionnalités du langage. Ce *design* le rend plus simple à adopter pour les utilisateurs, python étant un langage simple à utiliser et largement adopté par la communauté bioinformatique. Les workflows sont décrits dans les Snakefiles comme une suite de règles, les processeurs de Snakemake, qui sont reliés à l'exécution par leurs inputs et outputs. Les règles peuvent contenir des scripts en plusieurs langages, mais aussi des "shell" depuis lesquels il est notamment possible de lancer des outils ou, dans une moindre mesure, des services web. Cette approche rend Snakemake très flexible.

Il n'existe pas d'interface graphique pour Snakemake, les utilisateurs doivent coder les workflows. Même si cela nécessite plus de connaissances de programmation de la part de l'utilisateur, cela rend Snakemake plus flexible et polyvalent (dans la gestion dynamique des dépendances, par exemple). Snakemake permet également de visualiser les graphes d'exécution, ou DAG (Directed Acyclic Graph) des workflows.

Exécution du workflow.
Pour faciliter les reprises sur erreur des workflows, Snakemake n'exécute les règles que si les fichiers de sortie ne sont pas présents ou si la modification des fichiers d'entrée est plus récente. Il propose également des fonctionnalités de gestion des fichiers intermédiaires ou issus d'exécutions incomplètes, qui peuvent être supprimés automatiquement.

Contexte et environnement d'exécution.
Snakemake est portable, etpeut se lancer dans des environnements variés, sur des machines individuelles comme sur des infrastructures de calcul haute performance (HPC). Il intègre également des solutions de gestion des dépendances logicielles afin de favoriser la reproductibilité des workflows.

Nextflow. Nextflow [15], créé en 2017 par Di Tommaso et al, est un autre système de gestion de workflows ayant gagné en popularité en bioinformatique.

Spécification du workflow. De la même façon que Snakemake, les workflows Nextflow sont écrits dans un DSL, et doivent être codés par l'utilisateur, n'offrant pas d'interface graphique.

Les processeurs Nextflow, les process, peuvent comme pour Snakemake contenir des scripts ou des shell, et depuis 2020, des sous workflows. Ils sont reliés entre eux par des channel, canaux de flux de données explicites, contrairement à Snakemake où ils sont implicites. Il est possible de spécifier des opérateurs sur ces canaux, définissant ce que l'on veut faire d'un flux de données (le répliquer, le trier,...). C'est un modèle "top-down" qui suit le flux naturel de l'analyse des données. Ce modèle ne nécessite pas de calculer un arbre de dépendances entre les tâches, ni de préciser les fichiers que l'on souhaite avoir en sortie.
Même si Nextflow n'offre pas d'interface graphique, il est également possible de générer des visualisations des flux de données ou connections entre les processeurs des workflows.
Nextflow et Snakemake sont des langages déclaratifs, dans lesquels il est possible d'intégrer des fragments de code déclaratif (shell des processeurs par exemple). Cependant, il y a quelques différences dans l'implémentation des workflows.
Dans Nextflow, on définit le pipeline en décrivant les étapes ainsi que les dépendances entre les étapes - *les channels*. Nextflow s'occupe ensuite de la gestion des ressources et de l'exécution des étapes de manière parallèle et distribuée. Cela permet une abstraction plus élevée et une meilleure portabilité des pipelines, car Nextflow gère automatiquement les aspects liés à l'environnement d'exécution.
D'un autre côté, en Snakemake, l'accent est mis sur la description des étapes (*les règles*) et leurs entrées et sorties. Snakemake fonctionne en évaluant ces règles et en déterminant l'ordre d'exécution des étapes en fonction des dépendances entre elles. Bien que cela offre un contrôle plus fin sur le flux d'exécution, cela peut rendre le code plus verbeux et moins portable entre différentes configurations d'environnement.

Exécution du workflow.
Nextflow propose un mode de reprise sur erreur, qui déduit, à partir des canaux d'entrée et de sortie de chaque processeur, ainsi que des données initiales, où il doit redémarrer. Si un process est ajouté ou modifié, ou si les données d'entrée sont modifiées, seuls les calculs affectés par le changement sont ré-exécutés dans ce mode. De plus, il est possible de spécifier si un processeur doit être réexécuté à chaque fois ou non.

Contexte et environnement d'exécution.
Nextflow ne prend pas en charge directement la gestion des dépendances logicielles, mais est compatible avec des systèmes de gestion de dépendances, comme Snakemake.

Systèmes de gestion de workflows dans d'autres disciplines

Il existe une multitude d'autres systèmes de gestion de workflows. Dans cette section, nous présentons certains de ces systèmes notables qui ont été adoptés par la communauté bioinformatique, mais également largement par d'autres communautés. Parmi ces systèmes, nous examinerons en détail trois d'entre eux : Kepler, Vistrails et KNIME.
Il convient de noter que si les systèmes précédemment mentionnés étaient principalement conçus pour l'analyse de données, ce n'est pas le seul domaine d'application des systèmes de gestion de workflows. Nous décrirons un de ces systèmes, axé sur la modélisation des systèmes biologiques, OpenAlea.

Kepler, créé en 2004 par Altintas et al, puis repris par le projet Kepler avant d'être abandonné en 2016, est un système de gestion de workflows. Il est généraliste, bien qu'utilisé majoritairement par des communautés "sciences de la vie", il a également été adopté par les communautés de l'astronomie et de l'astrophysique. Kepler offre une interface conviviale qui permet aux utilisateurs de concevoir des workflows en utilisant une représentation visuelle plutôt que du code. Les workflows sont construits comme des modèles de diagrammes de flux de contrôle, en connectant des composants représentant des tâches (les acteurs) et des flux de données (les *channels*). Il propose une liste prédéfinie d'*acteurs* que les utilisateurs peuvent connecter pour créer des workflows, en les connectant grâce aux channels. Les utilisateurs peuvent écrire leurs propres acteurs mais cela nécessite généralement une expertise plus avancée en programmation. Les workflows de Kepler peuvent être échangés en XML.Kepler facilite le prototypage des workflows, pour que les utilisateurs puissent concevoir et tester leurs idées avant de les mettre en œuvre sous forme de code exécutable.
Il combine aussi de manière transparente la conception de workflows de haut niveau avec l'exécution et l'interaction en temps réel, l'accès aux données locales et distantes, ainsi que l'invocation de services locaux et distants.
Kepler n'intègre pas d'outils de gestion des dépendances externes. Cependant, il est axé sur la gestion de versions : la version d'un acteur est annotée dans la spécification du workflow, ce qui permet de conserver l'historique exact des workflows.

Vistrails, créé en 2005 par Bavoil *et al.*, est un système de gestion de workflows utilisé dans la bioinformatique, mais aussi la physique. Il propose une interface graphique conviviale qui permet aux utilisateurs de créer et d'exécuter des workflows en assemblant des modules préexistants en les reliant de manière visuelle. L'infrastructure de Vistrails lui permet de se combiner avec des systèmes et des

bibliothèques de visualisation existants [40]. Son composant clé, le "vistrail" (chemin de visualisation), est une spécification formelle d'un pipeline. Contrairement aux systèmes existants basés sur le flux de données, comme Kepler, dans VisTrails, il existe une séparation claire entre la spécification d'un pipeline et ses instances d'exécution. Les spécifications des Vistrails sont stockées dans un format XML, et c'est le premier système à proposer une fonctionnalité de gestion de versions spécifique à ce format.

KNIME, créé en 2006 par une équipe d'ingénieurs logiciels de l'université de Constance est le premier a avoir été conçu en tant que logiciel commercial. KNIME est utilisé non seulement en bioinformatique mais dans de nombreux domaines tels que la finance, la recherche en sciences sociales ou la santé. Il possède lui aussi une interface utilisateur graphique permet la construction de workflows par l'assemblage bout à bout de nœuds chacun réalisant une opération spécifique telle que le formatage des données, leur modélisation ainsi que l'analyse et la visualisation des résultats au sein de la même interface. KNIME inclut des plugins pour intégrer des outils d'analyse de données existants.

OpenAlea, créé en 2008 par une équipe Inria de Montpellier, s'adresse avant tout à la communauté de modélisation et à la simulation de systèmes biologiques et écologiques. Elle offre un ensemble d'outils et de bibliothèques pour la création de modèles, l'analyse de données et la visualisation dans le domaine des sciences du vivant. OpenAlea est codé en Python et offre également une interface graphique pour créer et manipuler les modèles et workflows.

Note sur les Business Workflows Les business workflows, en opposition aux workflows scientifiques, sont des workflows dont l'enchaînement des modules n'est pas uniquement basé sur le modèle *dataflow* mais sur le modèle de processus *control flow* [41]. L'exécution d'un processeur n'est pas nécessairement déclenchée par la fin de l'exécution du précédent (et par conséquent "l'arrivée" des données nécessaires en entrée), mais par des "décisions". Les décisions sont des points de contrôle, où des choix sont effectués en fonction de certaines conditions prédéfinies. Les flux de contrôle indiquent l'ordre séquentiel ou parallèle des activités. Les modules peuvent être des tâches manuelles (effectuées par des utilisateurs) ou automatisées (effectuées par des systèmes informatiques).
Un langage de modélisation de processus courant pour les business workflows est le Business Process Model and Notation (BPMN), pour représenter graphiquement

les business workflows via des diagrammes de processus.
Ces workflows sont ainsi principalement utilisés pour la gestion des ressources humaines, la gestion de projets, la gestion des ventes,... De ce fait, ils traitent généralement des données structurées et standardisées, comme des tableaux, des rapports, des formulaires, contrairement aux workflows scientifiques qui sont plus typiquement utilisés pour analyser et traiter des données hétérogènes et volumineuses.

Comparaison des trois systèmes utilisés en 2023

La principale différence entre Galaxy et les deux autres systèmes, Nextflow et Snakemake, réside dans les utilisateurs cibles. Galaxy cible les utilisateurs finaux sans compétences en programmation (le développement des workflows est réalisé par des actions de glisser-déposer sur des étapes prédéfinies). Ainsi les analystes de données ou experts du domaine peuvent eux même créer des workflows, en s'appuyant sur une aide de développeurs qui eux développeront les processeurs et l'infrastructure nécessaire.

En revanche, Nextflow et Snakemake ciblent les bioinformaticiens qui sont compétents dans les langages de script, les rendant de plus en plus populaires dans une communauté en constante évolution. Ces systèmes sont ainsi destinés à des développeurs de workflows qui peuvent eux mêmes conduire ou comprendre les analyses.

3.3 . Éléments de solution génériques

Parmi les éléments de solutions disponibles, la présente section introduit les approches qui ne sont pas dédiées à l'utilisation d'un langage ou d'un système spécifique pour représenter ou développer le pipeline. Les éléments de solutions présentées ici sont au contraire assez génériques pour être appliqués à des formes de pipelines variées.

La figure 3.1 [42], schématise ces solutions génériques qui permettent la gestion des pipelines. Afin d'adresser les défis liés à la reproductibilité et la réutilisabilité des analyses de données, on peut identifier quatre échelles distinctes où des éléments génériques de solution peuvent être appliqués.

1. Échelle des données : À ce niveau, il est essentiel de garantir que les données

utilisées dans l'analyse soient bien organisées, documentées et accessibles, et que des informations sur leur génération soit disponibles.

2. Échelle des pipelines : Pour assurer la reproductibilité et la réutilisabilité des pipelines, il est important d'adopter une approche modulaire plutôt que de créer des scripts linéaires. Les pipelines modulaires sont plus faciles à comprendre, à adapter et à partager. Une documentation détaillée sur la façon d'exécuter les pipelines est également cruciale.

3. Échelle de l'environnement logiciel : La gestion de l'environnement logiciel est essentielle pour garantir la cohérence et la stabilité des analyses. L'utilisation de gestionnaires de dépendances, de conteneurs logiciels ou d'environnements virtuels peut aider à créer des environnements reproductibles qui permettent de reproduire les résultats d'une analyse sur différentes machines.

4. Échelle de l'exécution : À ce niveau, il est important de définir clairement les paramètres d'exécution de l'analyse, tels que les valeurs de seuil, les paramètres d'optimisation, etc. Ces paramètres doivent être documentés de manière adéquate pour que les résultats puissent être reproduits avec précision.

En intégrant des éléments génériques de solution à chacun de ces niveaux, les chercheurs peuvent améliorer considérablement la reproductibilité et la réutilisabilité de leurs analyses de données, facilitant ainsi la collaboration.

3.3.1 . Aide au choix des outils

Les outils bioinformatiques sont indispensables dans les pipelines pour résoudre des tâches spécifiques, gérer la complexité des analyses, faciliter la représentation des étapes, assurer la reproductibilité, et automatiser les tâches, ce qui améliore l'efficacité et la fiabilité des analyses sur les données biologiques.
Bien choisir les outils bioinformatiques à utiliser dans un pipeline bioinformatique est donc une tâche particulièrement importante. Il existe une grande quantité d'outils en bioinformatique, leur nombre augmente chaque année [43]. Il n'est pas toujours simple de faire son choix parmi tous ces outils. En particulier, l'utilisateur doit faire face à la dispersion des sources décrivant les outils existants. Ces sources sont réparties sur de nombreuses pages web (GitHub ou les sites de documentation des outils par exemple), ou dans la littérature scientifique. Les moteurs de recherche classiques aident parfois à retrouver quelques outils, mais avec des résultats de qualité variable.

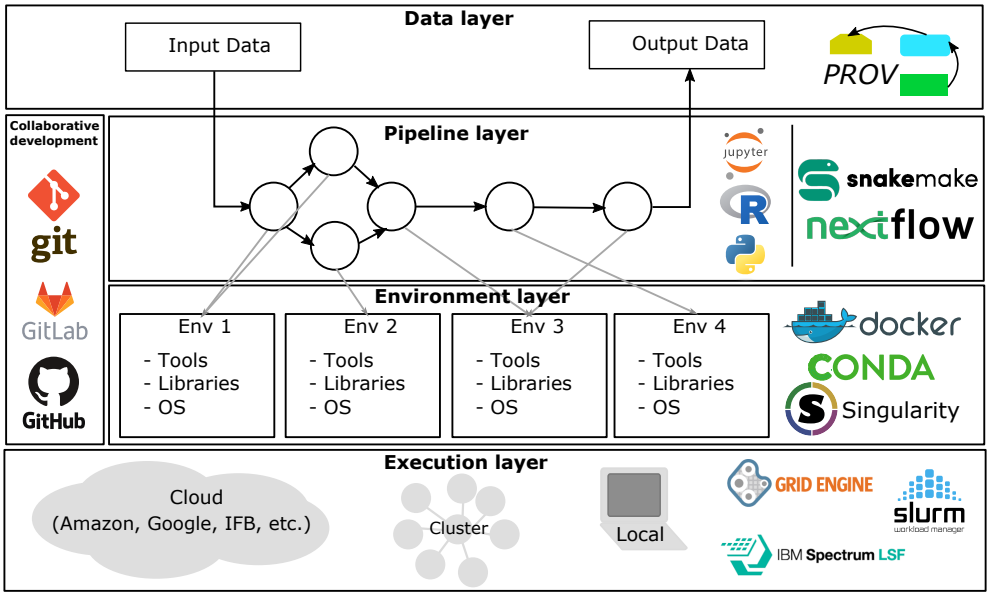

Figure 3.1 – Éléments génériques de solution. On peut y distinguer quatre échelles différentes : celle des données, des pipelines, de l'environnement logiciel et de l'exécution. À chaque niveau, on trouve certains éléments de solutions qui sont à ce jour largement utilisées.

Pour répondre à ces problématiques, plusieurs solutions pour recenser des outils bioinformatiques ont été développées. Nous en décrivons cinq ici.

Catalogues spécialisés

Parmi les solutions recensant les listes d'outils, on trouve notamment des sites pour lister et catégoriser des outils bioinformatiques, en particulier **ms-utils.org** et **expasy.org**.

ms-utils.org, a été créé en 2006 par Magnus Palmblad de l'université de Leiden. C'est un wiki qui liste environ 200 outils gratuits pour l'analyse de spectrométrie de masse, en les triant dans des catégories. Ce site est maintenu par une seule personne mais s'enrichit grâce aux suggestions des experts de la communauté. Il ne concerne qu'un domaine très restreint, et donc qu'un petit nombre d'outils, et son format "liste" n'a que l'avantage de centraliser l'information.

expasy.org est un catalogue d'outils plus large développés par le Swiss Institute of Bioinformatics. Il propose une liste plus conséquente que celle présentée précé-

demment (avec un total de 160 outils et bases de données). Une autre différence saillante avec la solution précédente est la capacité de cette ressource à indexer les outils grâce à des mot-clés. Enfin, elle est dotée d'un moteur de recherche.

Ces deux solutions demeurent des initiatives à petite échelle et, bien que particulièrement utiles pour des communautés ciblées, elles ne répondent qu'à une partie des problèmes cités. La conception de pipelines peut en effet concerner des domaines variés, ainsi la centralisation marginale de ces outils n'est qu'une solution partielle.

Plateformes de découverte d'outils généralistes

Les trois solutions que nous présentons maintenant contiennent un plus grand nombre d'outils et portent sur un spectre applicatif plus large.

La plateforme pionnière est **BioCatalogue** [44], créée en 2009 et co-développée par l'Université de Manchester et l'EMBL-EBI. Elle fut la première à fournir une liste organisée de services web bioinformatiques. Elle a proposé un large éventail de plus de 1400 services Web. Les services Web répondaient il y a 15 ans à des standards Web particuliers. Ces services étaient regroupés en catégories et accompagnés de métadonnées détaillées, telles que les descriptions, les formats d'entrée et de sortie, les liens vers la documentation et les exemples d'utilisation. En utilisant le Biocatalogue, les chercheurs et les scientifiques pouvaient trouver rapidement des services Web pertinents pour leurs besoins, évaluer leur qualité et leur pertinence, et intégrer ces services dans leurs workflows et analyses. Cela permet d'économiser du temps et des efforts dans la recherche et la sélection des outils bioinformatiques appropriés. Biocatalogue encourageait également la communauté à contribuer en soumettant de nouveaux services Web, en mettant à jour les informations existantes et en partageant leurs expériences et leurs évaluations. Cela favorisait la collaboration et l'échange de connaissances au sein de la communauté bioinformatique. Le projet n'est plus maintenu depuis 2014.

Quelques années plus tard, une initiative de centralisation "massive" d'outils bioinformatiques est aussi apparue, **Omic-tools** [45]. Cette plateforme, créée en 2012 par Arnaud Desfeux, visait à recenser et référencer plus de 19 200 outils[1] et bases

1. D'après https://www.re3data.org/repository/r3d100012426, consulté pour la dernière fois le 20/07/23

de données de bioinformatique. Ces outils sont utilisés pour analyser et interpréter les données génomiques, transcriptomiques, protéomiques et métabolomiques. Ils étaient annotés dans Omictools en utilisant un vocabulaire contrôlé propre à la plateforme. Les ressources présentes sur la plateforme ont été renseignées par la communauté. OmicTools devient en 2019 une plateforme commerciale, pour proposer aux chercheurs et aux entreprises des solutions payantes pour l'aide à la conception de protocoles d'analyse. Cette plateforme a cessé d'être maintenue en 2020.

En 2014, le réseau Européen de bioinformatique Elixir a introduit **bio.tools** [46]. Cette plateforme est la plus visible en 2023, elle offre un vaste référentiel d'outils bioinformatiques et comporte, en janvier 2023, un total de 27 538 outils.
Afin d'aider à la conception de pipelines bioinformatiques, les outils y sont annotés selon le modèle d'information formel **biotoolsSchema** [47]. biotoolSchema sert à décrire une large gamme de logiciels (des outils en ligne de commande, services web, des environnements de travail,*etc.*). Il permet de renseigner des informations à différents niveaux de granularité, mais vise avant tout à concilier la richesse des informations disponibles et la facilité d'adoption. Il permet par exemple d'annoter un outil avec le DOI de la publication associée, les auteurs de l'outil, *etc.* Ce modèle aide notamment à proposer un cadre standard pour décrire des outils via une variété d'informations "techniques" propres à leur fonctionnement.

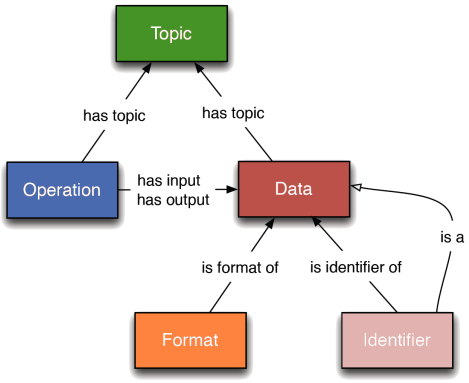

Figure 3.2 – Architecture de l'ontologie EDAM

Le fait d'accéder à un vocabulaire structuré et comportant un nombre important de termes joue un rôle fort dans la capacité à indexer et retrouver un outil.

C'est l'ontologie **EDAM** [48], créée en 2013 par Jon Ison *et al.*, un projet de la communauté Elixir, qui est utilisée pour annoter les outils. EDAM offre un très large spectre de termes et se compose de plusieurs sous-ontologies (cf Figure 3.2) : les tâches que l'outil effectue (Operations), les thématiques associées (Topic), les données d'entrée et de sortie de l'outil (Data, Format) et l'identification de ces informations (Identifier). Elle structure des termes pour décrire les ressources bioinformatiques telles que les outils, les pipelines et les bases de données. Ce sont surtout des bioinformaticiens qui l'ont mise au point au cours de nombreux workshops. Elle est donc conçue par et pour les bioinformaticiens, afin de les aider à trouver des ressources appropriées pour leurs expériences, ainsi qu'à les décrire et les annoter de manière cohérente pour faciliter la recherche et la réutilisation de ces ressources. Elle contient plus de 2000 termes couvrant un large éventail de sujets en bioinformatique, tels que l'analyse de séquences, la modélisation moléculaire, la classification des protéines et la visualisation des données. EDAM est largement utilisée dans le domaine de la bioinformatique pour faciliter la recherche, l'annotation et l'interopérabilité des ressources bioinformatiques. Par ailleurs, les classes EDAM, en plus d'être utilisées dans biotools, ont également été adoptées par expasy.org et ms-utils.org pour l'annotation et la classification de leurs outils.

Ces annotations permettent aux utilisateurs de bio.tools de les trouver facilement et donc de **réutiliser** les mêmes outils. La réutilisation est ancrée dans les pratiques des bioinformaticiens, et permet une plus grande fiabilité et consistance dans les algorithmes utilisés pour les analyses. C'est notamment grâce à ces informations, parfois très précises, que ces outils sont indexés, rendant la recherche via le moteur de recherche de bio.tools aisée.
Les entrées EDAM sont renseignées par un millier de contributeurs, qui sont membres de la communauté bioinformatique, mais aussi par ses développeurs qui peuvent avoir recours à des méthodes automatisées [2] pour la remplir.

En résumé, bio.tools est une plateforme d'ampleur, dont l'intérêt et le succès résident dans un effort partagé d'une grande partie de la communauté bioinformatique pour renseigner et maintenir cette plateforme centralisée d'outils et destinée à couvrir tous les domaines de la bioinformatique. Elle bénéficie également d'un modèle d'information ainsi que d'un vocabulaire contrôlé riche dédiés aux ressources logicielles bioinformatiques. Ces nombreuses caractéristiques en font à ce jour la plateforme de partage d'outils qui répond le mieux à la fois à la problématique de centralisation des informations, mais également à l'aide au choix d'outils en

2. https://github.com/bio-tools/pub2tools

fournissant un cadre structuré d'annotations aidant à la comparaison grâce à biotoolsSchema.

Les caractéristiques des différentes plateformes sont regroupées dans le tableau 3.1.

	Période d'activité	Auteurs et communautés	Domaines (1)	Nb d'outils
ms-utils.org	2006 – présent	Magnus Palmblad	*	200
expasy.org	2012 – présent	SIB	**	160
Biocatalogue	2009 – 2014	EMBL	***	1400
Omictools	2012 – 2020	Henry *et al.*	***	19200
bio.tools	2014 – présent	Elixir – EMBL	***	27538

Table 3.1 – Comparaison des plateformes d'indexation d'outils.
(1) : * : domaine de la spectrométrie de masse, ** : domaine généraliste provenant d'un seul institut, *** : domaine généraliste provenant de multiples sources

3.3.2 . Développement de pipelines

Pour écrire leurs pipelines, les bioinformaticiens exploitent un large spectre de logiciels ou plugins. Nous distinguerons les environnements de développement intégrés (IDE) et les plateformes de partage de code.

Environnement de développement

Lorsque les pipelines sont implémentés sous forme de simples scripts, en bash ou en python par exemple, différents environnements de développement intégrés (IDE) peuvent être utilisés. Ces IDE peuvent être "génériques", ou compatibles avec plusieurs langages. Les plus populaires sont Visual Studio Code, Sublime text, Vim et Emacs.

D'autres IDE sont dédiés à un langage spécifique. Des IDE de langages populaires en bioinformatique sont Spyder et PyCharm pour Python, Rstudio pour R, Matlab Editor pour Matlab et Eclipse pour Java. Ces IDE offrent, pour améliorer le confort des utilisateurs, la possibilité d'ajouter des plugins. Ces plugins peuvent par exemple faciliter l'intégration avec des bases de données publiques, ou d'outils de visualisation de molécules, de pré-traitement de données,... Par exemple, le plugin biomaRt [49] est un plugin R qui permet d'accéder facilement à des bases de données publiques comme Ensembl [3], UniProt [4], NCBI [5], et bien d'autres. Il offre

3. https://www.ensembl.org/index.html
4. https://www.uniprot.org/
5. https://www.ncbi.nlm.nih.gov/

une interface conviviale pour explorer la structure des bases de données, effectuer des requêtes complexes pour récupérer des informations spécifiques, et intégrer ces données dans leurs analyses.
Il existe également des environnement de développement dédiés pour les pipelines implémentés dans des notebooks.
L'IDE **Rstudio**[6] aide à visualiser et à compiler les notebooks en Sweave et knitR (R).
Depuis 2018, le projet Ipython propose un environnement de développement interactif pour les notebooks, **JupyterLab** [35]. L'interface s'ouvre, comme celle de Jupyter Notebook, dans un navigateur. Elle est composée d'une "zone de travail", qui peut comporter plusieurs onglets, d'une barre latérale gauche et d'un menu. La barre latérale gauche contient entre autres un navigateur de fichiers, la liste des noyaux et des terminaux en cours d'exécution et la liste des onglets.
La zone de travail principale de JupyterLab permet de disposer des documents (notebooks, scripts,*etc*) et terminaux dans des panneaux d'onglets qui peuvent être redimensionnés ou subdivisés. Une fonctionnalité intéressante de JupyterLab pour les développeurs de pipelines est ainsi la possibilité de créer des *vues* pour les résultats (intermédiaires ou non) des notebooks, ce qui facilite l'implémentation et test des pipelines. Les notebooks restent accessibles lors de la navigation dans les différentes parties du code. La présentation des résultats est également facilitée par des outils interactifs de visualisation des données.

L'interface est personnalisable et permet une organisation en *vues* qui permet de visualiser plusieurs documents de types différents (données, résultats intermédiaires, résultats, scripts,...) à la fois, ce qui la rend particulièrement adaptée à l'analyse de données. JupyterLab est modulaire et extensible, et il est possible d'y ajouter des plugins pour y intégrer de nouveaux éléments.
Bien que JupyterLab offre de nombreuses fonctionnalités avancées, il ne répond pas nécessairement de manière optimale aux besoins spécifiques de l'écriture de pipelines complexes et à l'intégration de plusieurs outils en ligne de commande dans le domaine de la bioinformatique. Les notebooks peuvent présenter des limitations lorsqu'il s'agit de gérer des pipelines plus élaborés, notamment en termes de gestion des dépendances entre les différentes étapes du pipeline et d'exécution non linéaire du code.

6. https://posit.co/products/open-source/rstudio/

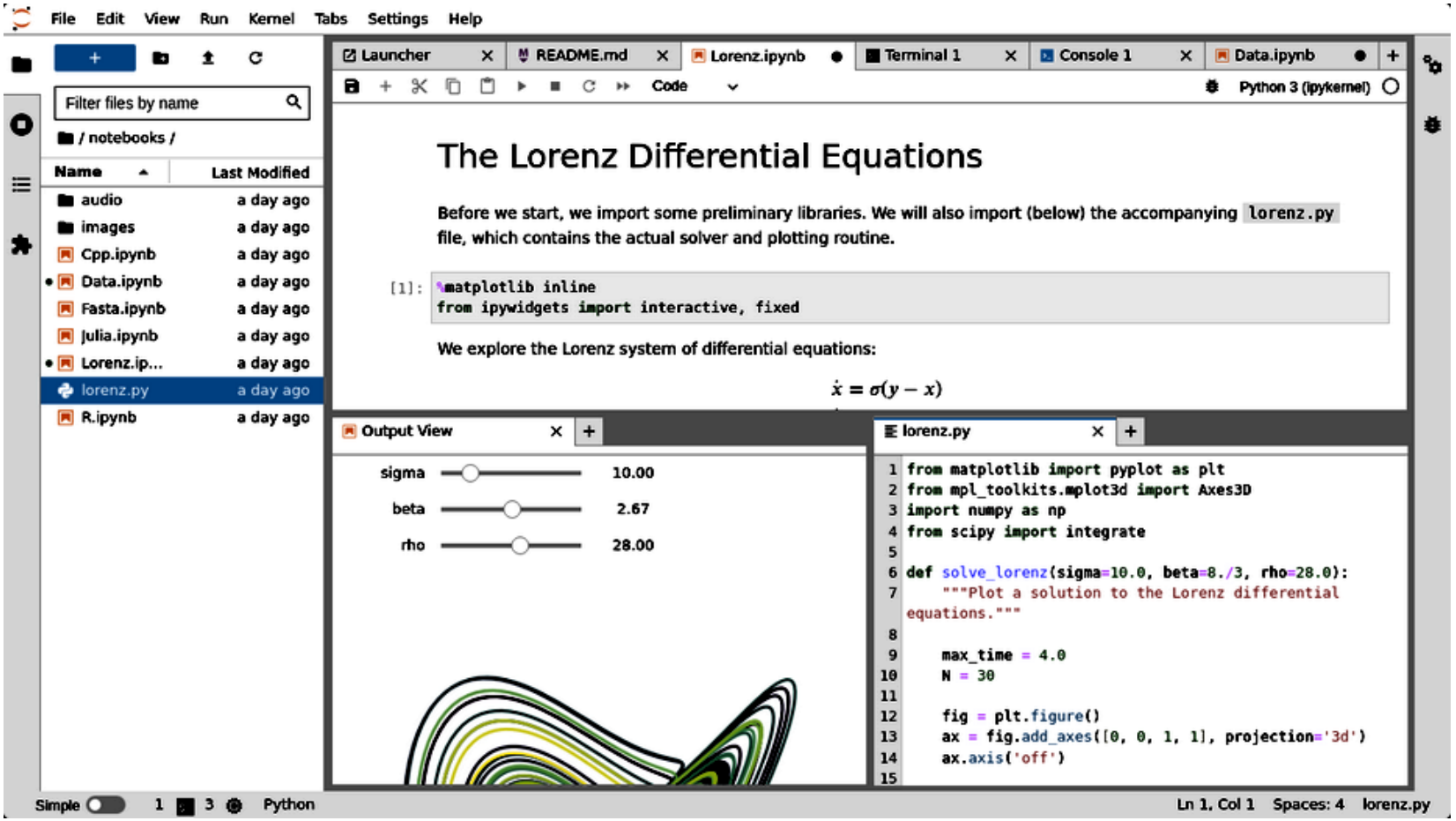

Figure 3.3 – Capture de l'interface Jupyterlab issue de la documentation de Jupyter Lab [a].

a. https://jupyterlab.readthedocs.io/en/latest/

Plateformes de partage de code collaboratives

Pour partager et examiner leur code et leurs pipelines, les bioinformaticiens travaillent sur des plateformes collaboratives.
Romano *et al.* [50] propose une revue (en 2011) sur les plateformes de partage utilisées par la communauté bioinformatique. On y retrouve **GitHub**[7], lancé en 2008 par Chris Wanstrath, PJ Hyett et Tom Preston-Werner et basé sur Git[8], logiciel de gestion de versions décentralisé libre créée en 2005 par Linus Torvalds. Les bioinformaticiens utilisent GitHub pour partager du code, pour collaborer sur des projets et pour contrôler les versions des codes sources. C'est un service d'hébergement de référence pour les répertoires Git. Un **dépôt** Git est un répertoire qui a été placé sous contrôle de version, contenant des fichiers ainsi que toutes les modifications suivies. Un **commit** est un instantané des modifications suivies qui est conservé dans le dépôt ; les développeurs créent des *commits* chaque fois qu'ils souhaitent conserver un instantané. GitHub propose de nombreuses fonctionnalités qui aident à la gestion du code, comme la gestion de versions, le **fork** de dépôts Git pour copier et modifier sa propre version du dépôt de quelqu'un d'autre, mais aussi des interactions sociales, comme le suivi d'utilisateurs ou de dépôts via la **souscription**. Romano *et al.* mentionnent également CodePlex[9], en activité de 2006 à 2017 ainsi qu'une plateforme dédiée aux workflows Taverna en permettaient la composition collaborative : **Confucius** [51]. Cette plateforme, anciennement Co-Taverna, permettait la composition collaborative de workflows scientifiques. Cet outil, intégré à l'environnemnt de travail Taverna a été créé en 2010 (et abandonné en 2016) par Jia Zhang et visait à répondre au besoin d'un outil de création de workflows collaboratifs qui aide les scientifiques de domaine à concevoir, composer, annoter, exécuter, surveiller et gérer des workflows scientifiques sur Internet en modes synchrone et asynchrone.

Aujourd'hui, les bioinformaticiens utilisent très majoritairement GitHub pour partager leur code. Près de 50% des logiciels publiés dans la revue Bioinformatics en 2017 ont leur code source hébergé sur GitHub, ce qui témoigne de la popularité de la plateforme dans le domaine de la bioinformatique [52].

Il existe des plateformes de développement collaboratives dédiées à l'écriture de pipelines. Plusieurs plateformes de ce type existent pour les notebooks. Un environnement basé sur les notebooks Jupyter est Google colab[10], un environnement

7. http://github.com/
8. https://git-scm.com/
9. http://www.codeplex.com/
10. https://colab.research.google.com/

de développement de Python hébergé sur Google Cloud [11]. Pour R, il existe RStudio Cloud [12].
À ce jour, la dernière plateforme collaborative dédiée à l'écriture de workflows issus d'un système de gestion de workflows est Galaxy.

3.3.3 . Traçage des données

Lorsqu'une première version du pipeline a été développée, la phase de test peut commencer et nécessite des jeux de données dédiés. Il est très important dès cette étape de garder la trace des données consommées et produites par le pipeline pour comprendre les résultats obtenus. L'échelle des "Données" définie dans la Figure 3.1 est relative à ce traçage des ensembles de données consommés et produits lors de l'exécution d'un pipeline de bioinformatique. Représenter et tracer la *provenance des données* de manière uniforme est essentiel, à la fois pour tester et documenter les pipelines, ce qui facilitera leur réutilisation.
Juliana Freire et Susan Davidson [53] ont introduit deux formes distinctes de provenance : la provenance prospective et la provenance rétrospective. La provenance prospective capture la spécification des pipelines, les étapes de l'analyse, les outils utilisés... La provenance rétrospective capture l'exécution des pipelines, les étapes qui ont été exécutées et les données utilisées et générées.

Ontologies pour représenter la provenance

Plusieurs ontologies ont été introduites pour représenter les informations de provenance [54]. Trois ontologies ont joué un rôle important : OPM (The Open Provenance Model), PROV-O (The Provenance Ontology) [13] et P-Plan (The Plan-Provided Provenance Ontology). Chacune de ces ontologies a été conçue avec des objectifs différents et a des avantages et des inconvénients en fonction de l'application pour laquelle elle est utilisée.

Le premier modèle de provenance est **OPM** [55] (Open Provenance Model). C'est un modèle standard pour la représentation de la provenance des workflows issu d'un effort de la communauté. Il a été créé en 2007 à l'occasion de provenance challenges dans le cadre des workshop de la conférence IPAW. C'est un modèle de provenance open-source pour la gestion des workflows scientifiques. Il est conçu spécifiquement pour les workflows. Son ontologie est OPMO (The Open Provenance Model OWL Ontology). OPM est conçu pour être compatible avec d'autres modèles de provenance.

11. https://cloud.google.com/?hl=fr
12. https://login.rstudio.cloud
13. https://www.w3.org/TR/prov-overview/

PROV a été standardisé par le World Wide Web Consortium (W3C) en 2013 en tant que recommandation pour un modèle standard et un langage formel pour représenter la provenance sur le Web. PROV fournit un ensemble de concepts et de relations qui peuvent être utilisés pour décrire l'origine et l'historique des artefacts numériques, ainsi que leurs relations avec d'autres artefacts et les agents qui les ont créés, utilisés ou modifiés. PROV vise à fournir un cadre très générique pour représenter la provenance dans différents domaines d'application [56], et est conçu pour être extensible. Son ontologie est PROV-O.
Comme PROV-O ne fournit pas explicitement tous les concepts nécessaires pour modéliser les pipelines ou leurs exécutions, plusieurs langages d'extension compatibles PROV coexistent : wfprov [14], prov-wf [57] et ProvONE [15] CWLProv [16] ainsi que P-Plan (The Provenance-Aware Workflow Planning System).

CWLProv est une extension de PROV-O, créé par Khan *et al.* en 2019. CWlProv est une extension de PROV-O qui permet de représenter la **provenance rétrospective** dans les workflows CWL. CWLProv étend les classes et les propriétés de PROV-O pour spécifier les détails spécifiques à CWL, tels que les étapes du workflow, les entrées et sorties, les outils utilisés, *etc.* En utilisant CWLProv, les workflows CWL peuvent être décrits de manière plus détaillée en termes de provenance, ce qui permet de mieux comprendre comment les résultats ont été obtenus et d'assurer la reproductibilité des analyses.
P-Plan, créé par un groupe dirigé par Yolanda Gil en 2007, est une ontologie compatible avec PROV-O qui ajoute des concepts nécessaires pour décrire les plans d'exécution des pipelines (**provenance prospective**). Dans ces concepts, on retrouve les étapes du workflow, les flux de données entre ces étapes, les paramètres et les valeurs de retour, ainsi que les dépendances entre les étapes. Son objectif principal est de modéliser les différentes étapes d'un pipeline, les dépendances entre ces étapes, ainsi que les entrées et les sorties associées à chaque étape, indépendamment du langage de spécification.

Encapsulation de données de recherche

Un autre élément de solution pour faciliter la gestion et le partage des données de recherche est RO-Crate [58], créé en 2019 par Sefton *et al.*, est une autre solution pour faciliter la gestion et le partage des données de recherche de manière structurée et portable. C'est **une norme et un modèle de métadonnées** qui

14. http://purl.org/wf4ever/wfprov
15. http://vcvcomputing.com/provone/provone.html
16. https://github.com/common-workflow-language/cwlprov/

permet d'encapsuler des *données de recherche* ou *objets de recherche* dans des conteneurs structurés. RO-crate utilise des formats de métadonnées standard et s'appuie sur le Web sémantique pour permettre une représentation plus riche et interconnectée des données scientifiques. Il utilise le format JSON-LD pour représenter les métadonnées des objets de recherche et leurs liens. L'utilisation de RO-Crate vise ainsi à résoudre les challenges liés à l'interopérabilité entre les plateformes et à faciliter la gestion des données de recherche en les rendant plus faciles à trouver, à réutiliser et à partager, tout en préservant l'intégrité et la provenance des données.
RO-Crate est conçu pour être utilisé avec des outils de gestion de données de recherche populaires tels que Dataverse, Figshare, Zenodo et DSpace, et peut être intégré dans des workflows de gestion de données de recherche existants.
En utilisant RO-Crate, les chercheurs peuvent fournir des métadonnées riches et détaillées sur leurs données, faciliter la découverte de ces données par d'autres chercheurs, et permettre la réutilisation et la reproduction des résultats de recherche.

3.3.4 . Déploiement de pipelines

La quantité de données à analyser en bioinformatique a fortement augmenté ces dernières décennies, notamment grâce à l'avènement de technologies de séquençage à haut débit. Les approches classiques d'analyse de données ne suffisent plus à satisfaire les exigences de rapidité dans l'exécution des tâches d'analyse de données en sciences de la vie. Cela est principalement dû au coût élevé en ressources de calcul, rendant le déploiement de pipelines en local (avec une puissance de calcul et de mémoire limitées) de plus en plus inadapté.

Pour exécuter leurs pipelines, les biologistes et les informaticiens sont donc amenés à utiliser des plateformes HPC (High Performance Computing). Ces plateformes correspondent à différents types d'infrastructures de calcul distribué [59].

- Les **clusters**, des groupes de serveurs connectés en réseau pour fournir des capacités de calcul distribué ;
- Les **grilles de calcul**, des réseaux de ressources informatiques partagées entre différentes organisations pour exécuter des tâches de calcul intensives, pouvant utiliser des ressources informatiques hétérogènes ;
- Les **clouds**, des services de calcul haute performance, permettant aux utilisateurs de réserver (pour des clouds académiques) ou de louer (pour des clouds commerciaux) des ressources de calcul à la demande.

Ces infrastructures, bien que différentes, restent similaires sur leur principe. Elles sont typiquement composées de 4 éléments principaux :

- des **noeuds de calcul**, qui sont des serveurs (souvent de haute performance avec des processeurs multi-cœurs, une grande quantité de mémoire et un système de stockage rapide) ;
- du **stockage**, par exemple des disques durs, des disques SSD, des bandes magnétiques,...
- le **réseau**, pour connecter entre eux les différents composants ;
- une **couche logicielle** pour gérer et orchestrer les tâches sur le système (planification des tâches, la gestion de la mémoire, le contrôle d'accès et la sécurité).

Pour communiquer avec ces éléments et lancer un pipeline sur un cluster, il faut en général se connecter à un noeud "*login*" [29], depuis lequel on ne lancera pas de calcul. Il faut en revanche lancer un script ou une commande de soumission qui contient les informations sur le pipeline à exécuter ainsi que les ressources nécessaires pour son exécution, telles que le nombre de processeurs, la quantité de mémoire, le temps d'exécution maximum, etc. Ce script dépendra de l'ordonnanceur disponible. Ce sont les **ordonnanceurs**, aussi appelés "ordonnanceurs de tâches" ou "systèmes de gestion de ressources", qui planifient l'exécution des tâches soumises par les utilisateurs en prenant en compte les ressources disponibles, les dépendances entre les tâches et les politiques d'allocation de ressources, dans l'objectif d'optimiser l'utilisation des ressources [60]. Ces ordonnanceurs vont ensuite communiquer avec des noeuds d'exécution qui vont lancer les calculs, et les données intermédiaires et finales seront ensuite envoyées dans les noeuds de stockage.

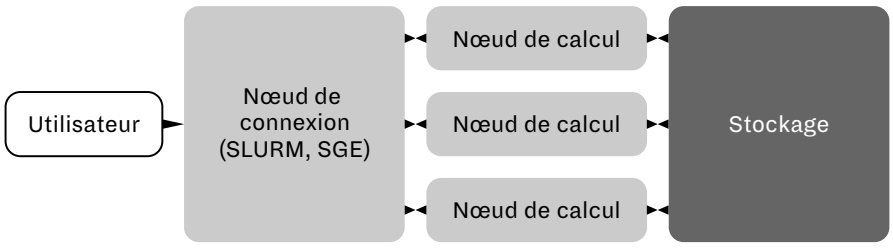

Figure 3.4 – Schéma d'architecture type de cluster

Les environnements de cluster HPC sont très différents, non seulement au niveau matériel (nombre de noeuds, forme de stockage,...) comme évoqué précédemment, mais également au niveau logiciel. Différents **ordonnanceurs** sont disponibles *e.g.*, SLURM [61], PBS [62], LSF ou SGE, et les scripts et commandes de lancement de tâches sont propres à chacun d'entre eux.
Lancer des pipelines "scripts" demande ainsi un travail considérable et une bonne connaissance de l'environnement HPC, et le portage à un autre environnement peut se révéler complexe.
Il est également possible de lancer des notebooks Jupyter dans des environnements HPC, et il est même possible d'interagir avec les notebooks. Leur lancement, même s'il peut nécessiter des commandes spécifiques ou des scripts simples, est plus aisé à mettre en place que le lancement d'un pipeline en bash par exemple. Le lancement interactif simplifie la démarche d'écriture ou de modification du pipeline, qui peut se faire en lançant les calculs "en direct".

3.3.5 . Maintenance et reproductiblité

La mise à disposition du code source d'un pipeline n'est pas suffisante pour qu'il soit reproductible. En effet, l'environnement logiciel des pipelines, qui font souvent appel à plusieurs outils et librairies, est une composante cruciale pour les réexecuter, et en reproduire les résultats. La capture, et surtout la mise à disposition des dépendances logicielles sont essentielles pour assurer la maintenance des pipelines et donc leur reproductibilité, ainsi que leur capacité à continuer de fonctionner dans le temps, et les rendre réutilisables par des tiers. Les défis techniques liés à cette capture comprennent les problèmes liés aux dépendances logicielles, à la documentation imprécise, à l'évolution du code et aux barrières à l'adoption de solutions existantes [63]. Des challenges majeurs sont de faire face aux problèmes suivants.

- **"Dependency Hell"** : Les dépendances logicielles posent souvent des problèmes lors de l'installation et de la construction du code, ce qui rend difficile la recréation de l'environnement de calcul d'origine.
- **"Code rot"** : Les dépendances logicielles évoluent régulièrement, ce qui peut modifier les résultats générés par le code. Les chercheurs doivent s'assurer que les résultats restent robustes face à ces changements.

L' "échelle environnement" de la Figure 3.1 présente plusieurs approches qui ont été proposées pour décrire, stocker et partager l'environnement d'exécution du pipeline et son contexte scientifique.

Trois familles de solutions coexistent : les hyperviseurs, les conteneurs et les systèmes de gestion de paquets.

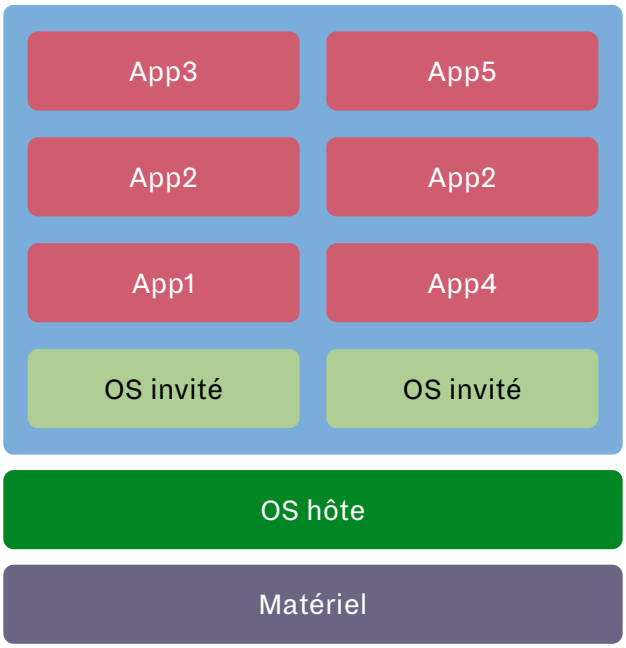

Figure 3.5 – Schéma d'architecture d'hyperviseur. L'hyperviseur est en bleu.

Les hyperviseurs

Tout d'abord, on distingue les technologies de virtualisation, ou **hyperviseurs** 3.5. Ces solutions, aussi appelées machines virtuelles, permettent l'abstraction au niveau du matériel. Chaque composant matériel nécessaire à l'exécution des logiciels doit être émulé par l'hyperviseur [64]. Il est possible d'accéder au véritable matériel via une interface fournie par l'hyperviseur, par exemple pour accéder à des fichiers sur un périphérique physique tel qu'un CD ou une clé USB, ou pour communiquer avec le réseau. À l'intérieur de cet ordinateur virtuel, un système d'exploitation et des logiciels peuvent être installés et utilisés comme sur n'importe quel ordinateur normal. Le matériel exécutant l'hyperviseur est appelé l'hôte (et le système d'exploitation hôte) tandis que toutes les machines émulées s'exécutant à l'intérieur sont appelées des invitées et leurs systèmes d'exploitation sont appelés des systèmes d'exploitation invités.

Il est possible d'installer une grande variété de systèmes d'exploitation sur les hyperviseurs, de manière assez indépendante du système d'exploitation hôte, les machines virtuelles étant très compartimentées du système hôte. Cette approche,

même si elle permet un grand contrôle de l'environnement logiciel, est plus lourde : le démarrage des machines virtuelles est lent, elles peuvent être gourmandes en ressources de calcul. Comme elles nécessitent le stockage et l'exécution de l'ensemble de l'environnement d'exécution (y compris le système d'exploitation), les fichiers résultant de l'export des machines virtuelles peuvent avoir une empreinte de stockage importante [65], ce qui complexifie leur partage. Par ailleurs, les machines virtuelles sont configurées "sur mesure" pour chaque environnement et la transposer à un autre environnement peut nécessiter des efforts supplémentaires.

Il est également possible de lancer plusieurs machines virtuelles en même temps. Elles peuvent se lancer sur des machines individuelles, ou sur des infrastructures de calcul HPC. Les hyperviseurs de type 1 sont donc plutôt destinés aux infrastructures de calculs. Des exemples d'hyperviseurs connus sont les suivants [66].

- **VMware Workstation** [17], créé en 1999 par VMWare company, et qui est propriétaire ;
- **KVM** [18], lancé en 2006 par Open virtualization alliance (OVA), logiciel libre et désormais intégré au noyau linux ;
- **Xen** [19], créé en 2005 par Paul Barham et al, open source ;
- **Vagrant** [20], créé en 2010 par Mitchell Hashimoto et John Bender, libre et open-source, qui peut être considéré comme un wrapper autour de logiciels de virtualisation ;
- **Virtualbox** [21], créé en 2007 par Oracle, libre.

Les conteneurs

Une alternative plus légère aux hyperviseurs sont les conteneurs 3.6. Tout comme les machines virtuelles, les conteneurs encapsulent les composants du système d'exploitation, les scripts, le code et les données dans un seul *package*, qui peut être partagé avec d'autres utilisateurs [65].

La virtualisation basée sur les conteneurs n'émule pas un ordinateur - pas d'émulation du matériel, contrairement aux hyperviseurs. Les conteneurs n'ont pas

17. https://www.vmware.com/
18. https://www.linux-kvm.org
19. www.xenrpoject.org
20. www.vagrantup.com
21. www.virtualbox.org/browser/vbox/trunk

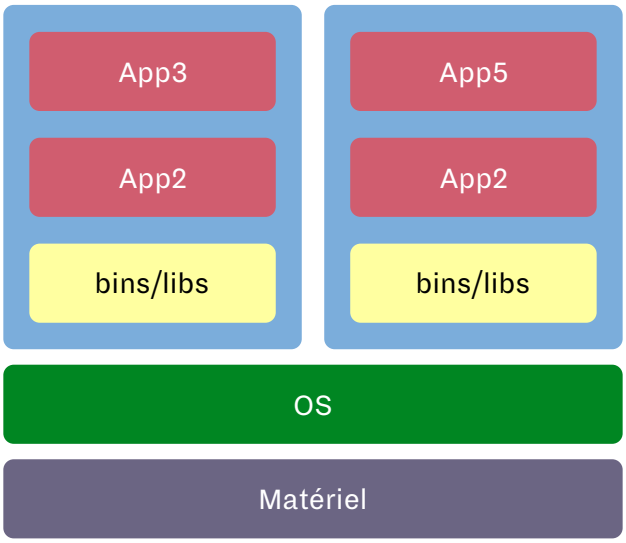

Figure 3.6 – Schéma d'architecture de conteneurs. Le conteneur est en bleu sur l'image.

de système d'exploitation invité, le système d'exploitation hôte fournit certaines fonctionnalités au conteneur [66]. Étant donné qu'il n'y a pas d'émulation complète du matériel, le logiciel s'exécutant dans les conteneurs doit être compatible avec le noyau et l'architecture CPU du système hôte. Les conteneurs sont ainsi limités dans les systèmes d'exploitations virtualisés, qui doivent être similaires au système hôte (ou des étapes supplémentaires, comme faire tourner le conteneur dans une machine virtuelle, doivent être adoptées). Le noyau Linux offre de nombreuses capacités pour isoler les processus. Il est donc à la base d'un grand nombre de solutions de conteneurs. Ces conteneurs ne capturent que les dépendances spécifiques requises par les applications et partagent les composants de bas niveau fournis par le système d'exploitation.

Les conteneurs sont construits à partir de *recettes*, des fichiers texte simples décrivant leur construction, et facilitant leur composition, leur gestion et leur partage.

En effet, les développeurs n'ont pas à configurer manuellement leurs machines, mais plutôt à distribuer la recette contenant tous les outils et configurations nécessaires et faisant quelques dizaines de kilo-octets et facilement transposable à d'autres machines et plateformes HPC. Cette approche facilite la maintenance et la gestion de logiciels et de versions car si l'on veut modifier le conteneur, il suffit de le re-générer et de le repartager.

À l'instar des hyperviseurs, il est possible de lancer plusieurs conteneurs à la fois.

Il existe deux approches de virtualisation pour les conteneurs. La première approche est la virtualisation au niveau du système d'exploitation. Les conteneurs partagent le même noyau Linux de l'hôte, ce qui permet une utilisation plus efficace des ressources système et une réduction des coûts de calcul liés à l'émulation matérielle des hyperviseurs. Une technologie courante de cette catégorie est OpenVZ [22], créée en 2005 par Parallels. C'est une technologie open source de virtualisation basée sur le noyau Linux. Les conteneurs OpenVZ offrent une isolation complète des ressources comme le processeur, la mémoire, le stockage et le réseau, et entre les différentes instances de conteneurs exécutées sur une même machine.

Une autre approche de virtualisation de conteneurs, plus flexible, est l'approche de conteneurisation. Une des technologies pionnières de cette approche est LXC [23]. Sortie en 2008 et basée sur le noyau Linux, elle a été développée par la communauté et par la suite intégrée au noyau. LXC fournit une isolation légère des conteneurs en utilisant les *namespaces*, les systèmes de fichiers et les utilisateurs. Cela permet une séparation plus stricte des ressources entre les conteneurs, tels que les processus, les systèmes de fichiers, les utilisateurs, la mémoire ou encore le CPU [67].

Des solutions largement adoptées en bioinformatique sont **Docker** [68] et **Singularity** [69].

Docker. Docker, créé en 2013 par Docker inc, est basé sur LXC. Il est compatible avec plus de kernels Linux et possède plus d'options pour gérer les données et les process, et fournit une meilleure isolation de système d'exploitation hôte [70]. Les conteneurs dockers sont créés à partir des images (elles mêmes créées depuis des recettes), et peuvent être très sommaires, ne contenant que des bases de systèmes d'exploitation, ou très sophistiquées, contenant un grand nombre d'applications. Il est possible de lancer des commandes manuellement ou automatiquement depuis des conteneurs.

Les conteneurs Docker sont simples à créer, ce qui est une force de cette technologie [64].

22. https://openvz.org/
23. https://linuxcontainers.org/

Les trois composants principaux de l'environnement Docker sont les suivants.

- Le **démon Docker**, appelé **dockerd**, est un processus persistant qui gère les conteneurs Docker. Le démon écoute les demandes envoyées via l'API **Docker Engine**. Le programme client Docker, appelé **docker**, fournit une interface en ligne de commande (CLI) permettant aux utilisateurs d'interagir avec les démons Docker ;

- Les **objets Docker** sont différentes entités utilisées pour assembler une application dans Docker. Les principales classes d'objets Docker sont les images, les conteneurs et les services ;

- Un **"Docker registry"** est une plateforme de stockage et de gestion pour les images Docker. Les clients Docker se connectent aux registres pour télécharger des images à utiliser ou téléverser des images qu'ils ont construites. Les registres peuvent être publics ou privés.

L'un des principaux facteurs qui empêche Docker de devenir la technologie de conteneur standard en HPC est lié à ses problèmes de sécurité [69]. Du point de vue de la sécurité informatique, une machine peut être considérée comme compromise si un utilisateur est en mesure d'exécuter du code en tant qu'utilisateur *root*. Si chaque utilisateur est en mesure d'obtenir le statut d'administrateur système, un utilisateur appelé "*root*", cela entraînerait des risques de sécurité inacceptables dans un environnement informatique partagé. Bien que cela ne pose pas de problème pour une utilisation en entreprise de Docker, car l'administrateur système sait quel code sera exécuté et quelle commande est utilisée pour invoquer Docker, un administrateur système d'un centre HPC n'a pas le même niveau de contrôle sur le code exécuté. L'un des principaux défis de la gestion d'un centre HPC dédié à la recherche est de permettre à chaque utilisateur d'exécuter du code sans passer par l'administrateur système, tout en garantissant simultanément que le système n'est pas compromis par un code malveillant (qu'il soit intentionnellement malveillant ou non).

Par ailleurs, de multiples problèmes de sécurité et de confiance peuvent découler de l'esprit de partage des images [64]. Pendant longtemps, Docker n'a pas disposé de mécanismes suffisants pour vérifier l'intégrité et l'authenticité des images partagées. Au contraire, leur système exposait des vulnérabilités et il était possible pour des attaquants de modifier les images et de contourner les mécanismes de vérification de Docker. Docker a abordé ces problèmes en intégrant un nouveau mécanisme appelé Content Trust qui permet aux utilisateurs de vérifier l'éditeur des images.

Singularity. Singularity, créé en 2017 par Kurtzer et al, propose une autre approche de conteneurisation. Cette méthode ne nécessite pas de privilèges root pour utiliser toutes les ressources, ce qui le rend déployable dans les systèmes HPC. Par ailleurs, il est compatible avec Docker, il est donc possible d'importer des images Docker dans Singularity. Comme mentionné précédemment, les conteneurs Singularity utilisent un seul fichier qui représente l'ensemble des fichiers présents dans le conteneur, contrairement à Docker qui utilise un format d'image basé sur des couches (*layer-based*), où chaque couche représente un ensemble de modifications apportées à l'image de base.
Un autre aspect fondamental de la reproduction des recherches est la préservation et la validation des données, et Singularity dispose d'une fonctionnalité en cours de développement qui garantira la validation du contenu du conteneur. Grâce à l'intégration directe du hachage SHA256, Singularity fournit une méthode de validation du conteneur qui garantit qu'une image de conteneur partagée n'a pas été modifiée ou altérée.

Partage d'images Docker. Les images Docker étant bien plus simples à partager que les machines virtuelles, grâce à leur taille et à la présence de l'application "Docker Registry", qui permet de stocker et distribuer des images Docker, mais propose aussi des fonctionnalités de gestion des versions, de collaboration et de déploiement automatisé pour faciliter le processus de développement et de déploiement des applications conteneurisées. L'exemple le plus populaire de plateforme de partage d'images publiques est Docker Hub[24]. Cette plateforme, lancée par Docker inc en 2014, contient une vaste collection d'images de conteneurs prêtes à l'emploi.
Docker Hub contient deux types de référentiels publics : officiels et communautaires. Les référentiels officiels contiennent des images publiques certifiées provenant de fournisseurs (par exemple, Canonical, Oracle, Red Hat et Docker). En revanche, les référentiels communautaires peuvent être créés par n'importe quel utilisateur ou organisation.
Cependant, puisque les images peuvent être ajoutées par n'importe qui, elles n'ont pas de critère de qualité. L'un des plus grands problèmes de Docker Hub est qu'il est difficile de faire face au grand nombre d'images contenant des vulnérabilités de sécurité [64].
L'origine de ce problème est que les images sur le Docker Hub ne sont pas mises à jour automatiquement, chaque mise à jour doit être réalisée manuellement. Les utilisateurs ont tendance à ne pas fréquemment mettre à jour leurs logiciels ou les conteneurs qu'ils ont partagés sur le Docker Hub [71].

24. https://hub.docker.com/

Une plateforme de partage d'images publiques majeure centrée autour de la bioinformatique est le projet **BioShaDock** [72], créé en 2015 par GenOuest core facility et l'IFB. Il est une réponse à la généricité de Docker Hub qui rend difficile pour un utilisateur de bioinformatique de trouver les images dont il a besoin. BioShaDock apporte plusieurs améliorations par rapport au registre Docker de base en termes d'authentification et de gestion des permissions, ce qui permet son intégration dans les infrastructures bioinformatiques existantes telles que les plateformes de calcul. Les métadonnées associées aux images enregistrées sont centrées sur le domaine, comprenant par exemple des concepts définis dans l'ontologie EDAM, un vocabulaire partagé et structuré de termes couramment utilisés en bioinformatique. Il ajoute également des constructions automatiques basées sur un Dockerfile ou un dépôt git. Les conteneurs et le registre sont dédiés à améliorer la reproductibilité des expériences bioinformatiques.

Le projet BioShaDock a été interrompu pour être fusionné avec le projet **BioContainers** [73, 74], lancé par Elixir en 2017. Ce projet vise à packager et mettre à disposition des outils bioinformatiques, de façon reproductible, et à les maintenir et mettre à disposition de la communauté. Il contient à ce jour 272.3K conteneurs et packages. La communauté BioContainers a développé un ensemble de directives pour standardiser les conteneurs logiciels, y compris les métadonnées, les versions, les licences et les dépendances logicielles. BioContainers prend en charge plusieurs technologies de packaging et de conteneurisation, comme par exemple Docker et Singularity. Sa structure repose sur deux composants principaux : (*i*) une organisation GitHub [25] comprenant tous les Dockerfiles (pour les conteneurs basés sur les fichiers Dockerfile), les spécifications et les outils pour créer/gérer les conteneurs ; (*ii*) les registres où les conteneurs disponibles sont construits par un système automatique et mis à disposition, prêts à l'emploi, notamment via Docker.

Les conteneurs ne peuvent être ajoutés que par la communauté BioContainers, mais les utilisateurs peuvent demander la mise à disposition d'un conteneur logiciel en ouvrant un *ticket* dans le dépôt GitHub du conteneur et en fournissant des informations sur le logiciel (nom, URL ou binaire à empaqueter).

Le **BioContainers Registry** [26] fournit une interface conviviale permettant aux utilisateurs de rechercher et de récupérer les outils bioinformatiques et les conteneurs correspondants. La page de recherche permet aux utilisateurs de rechercher un outil en utilisant des mots-clés, les packages étant annotés avec les annotations

25. github.com/BioContainers/
26. https://biocontainers.pro/registry

EDAM de bio.tools.

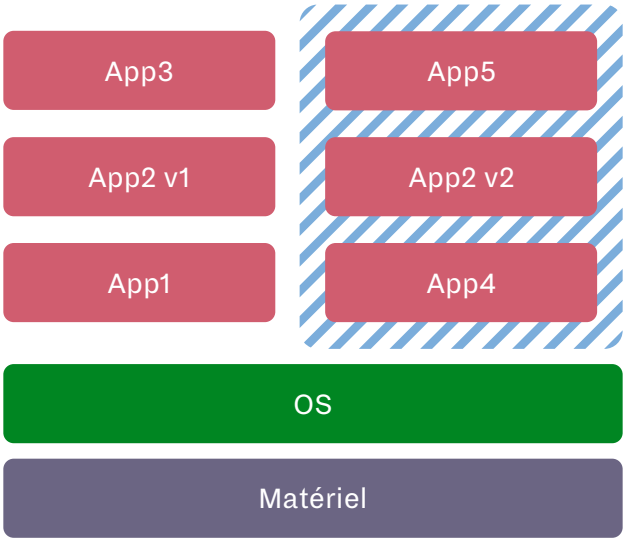

Figure 3.7 – Schéma d'architecture d'un environnement Conda. L'environnement Conda est en bleu sur la figure.

Systèmes de gestion de paquets et d'environnement

Le troisième type de solutions repose sur des systèmes de gestion de paquets et d'environnement, tels que **Conda** 3.7[27]. Ils facilitent l'installation d'outils et de dépendances, la gestion de l'environnement et le partage (dans une certaine mesure), mais ne sont pas complètement robustes face à l'hétérogénéité des machines d'exécution et des systèmes d'exploitation. Conda est un gestionnaire de paquets multiplateforme, donc indépendant du système d'exploitation. Écrit en Python, il peut être utilisé pour installer des logiciels écrits dans n'importe quel langage et des librairies python. Conda permet la création d'environnements distincts pour déployer plusieurs versions de paquets sur la même machine, en offrant une isolation légère, soit moins compartimentée que les conteneurs et les hyperviseurs. Conda ne nécessite pas de privilèges administratifs. Ces fonctionnalités rendent Conda adapté à une utilisation sur des environnements HPC, car le contrôle sur l'environnement d'exécution ne dépend pas de la configuration ou de l'accès au niveau du système. Un seul environnement Conda peut être lancé à la fois, et l'activation et désactivation de ces environnements n'est pas toujours simple à gérer ou à orchestrer. Par exemple, Conda modifie le chemin d'accès et les variables d'environnement

27. http://Conda.pydata.org

pour que les packages de l'environnement actif soient accessibles. Cela peut rendre complexe la gestion de plusieurs environnements actifs en même temps. Des divergences de versions de paquets entre environnements peut également être source de conflits de dépendances lors de l'activation et de la désactivation d'environnements. Cependant, la création et la maintenance des définitions de paquets Conda (appelées recettes) ne nécessitent aucune connaissance en programmation en dehors du *scripting shell* de base. Cet avantage a conduit à une adoption rapide de Conda par la communauté scientifique [32].
Le partage de ces environnements est très simple, ils peuvent être exportés et stockés dans un fichier yaml.
Un autre système de gestion de paquets et d'environnement populaire est Guix [28], soutenu par la Free Software Foundation (FSF). Guix propose un système de gestion de paquets fonctionnels, ce qui signifie que plusieurs versions de chaque paquet peut être installé "côte à côte" sans qu'ils n'interfèrent les uns avec les autres [75]. Guix favorise la reproductibilité et la transparence des environnements logiciels : contrairement à Conda, Guix ne repose pas sur la mise à disposition de binaires pré-compilés mais les instructions d'installation de paquet contiennent toutes les instructions nécessaires pour compiler chaque logiciel de façon déterministe. Les environnements Guix peuvent être plus légers en termes de taille, car ils incluent uniquement les dépendances nécessaires pour exécuter des applications spécifiques. Cependant, il est moins facile d'utilisation que Conda, et malgré de nombreuses initiatives de la communauté, Conda bénéficie d'un écosystème plus large et diversifié de paquets.

28. https://guix.gnu.org/

3.4 . Solutions offertes par les systèmes de workflows

Les systèmes de workflows scientifiques jouent un rôle central dans l'orchestration de toutes les couches décrites dans la figure 3.1, et agissent à toutes les étapes du cycle de vie des pipelines.
Nous passerons en revue les systèmes de workflows scientifiques, en particulier Snakemake et Nextflow, en examinant leur capacité à fournir des solutions à chaque étape du cycle de vie des pipelines.

3.4.1 . Développement

Le système de workflows populaire qui bénéficie d'un environnement de développement interactif est Galaxy [13]. L'éditeur de workflows s'ouvre sur un navigateur : comme évoqué précédemment, Galaxy est un serveur web, et les instances de Galaxy peuvent être publiques, privées ou locales. Sur l'interface web du serveur public de Galaxy [29], on retrouve différents outils de gestion de données, d'analyse de données et d'exploration de l'historique. Sur l'interface de création de workflows 3.8, la fenêtre centrale permet de "dessiner" les workflows. Les opérations sont encapsulées dans des outils répertoriés dans un *Toolshed* (la boîte à outils de Galaxy). Ces outils sont situés sur la barre latérale gauche de l'interface, et les utilisateurs peuvent les glisser-déposer dans l'espace de composition et les relier entre eux par des flux de données. Les paramètres de ces outils peuvent être changés sur le panneau des paramètres. La barre latérale droite contient l'**historique d'analyse**. La fonctionnalité historique d'analyse de Galaxy permet aux utilisateurs de conserver un enregistrement détaillé de toutes les étapes et résultats de leurs analyses, facilitant ainsi la traçabilité, la reproductibilité et le partage des données et résultats. Cette fonctionnalité offre une vue chronologique des manipulations effectuées, permettant aux utilisateurs de ré-exécuter des étapes spécifiques ou de gérer plusieurs versions d'une analyse enregistrée. Galaxy dispose également d'une gamme d'outils de visualisation qui aident les utilisateurs à explorer leurs données. L'environnement et ses fonctionnalités sont spécialement conçus pour permettre aux utilisateurs de créer des workflows sans avoir besoin de connaissances en programmation.
Le Toolshed est une fonctionnalité de Galaxy qui permet de dissocier l'implémentation du workflow et l'implémentation des tâches du workflow. Pour intégrer une tâche à un workflow, que ce soit une étape "simple" de manipulation de données (par exemple l'ajout d'une valeur dans un tableau) ou un outil bioinformatique, il faut qu'elle soit encapsulée dans un *outil* disponible sur le *Toolshed*. L'implémentation d'outils du Toolshed est moins triviale et nécessite des connaissances de

29. https://usegalaxy.org/

programmation.
Pour la création de workflows Snakemake et Nextflow en revanche, il n'existe pas d'environnement dédié. Il peuvent être écrits dans de simples éditeurs texte.
Cependant, des plugins pour des IDE populaires existent pour améliorer le confort de développement des utilisateurs. Pour Nextflow, des plugins ont été créés par une communauté de créateurs de workflows Nextflow, nf-core [76], pour Visual Studio Code et Atom. D'autres membres de la communauté Nextflow en ont écrit pour d'autres IDE : Sublime text, Emacs et Vim.
Un plugin "officiel" de l'équipe Snakemake existe pour l'IDE Pycharm.
Ces plugins proposent des fonctionnalités comme la coloration des éléments de syntaxe des langages, la mise en valeur des élements des processeurs, l'autocomplétion pour les variables spécifiques, des visualisation de la structure du code, le repli des blocs de code des processeurs, *etc.*

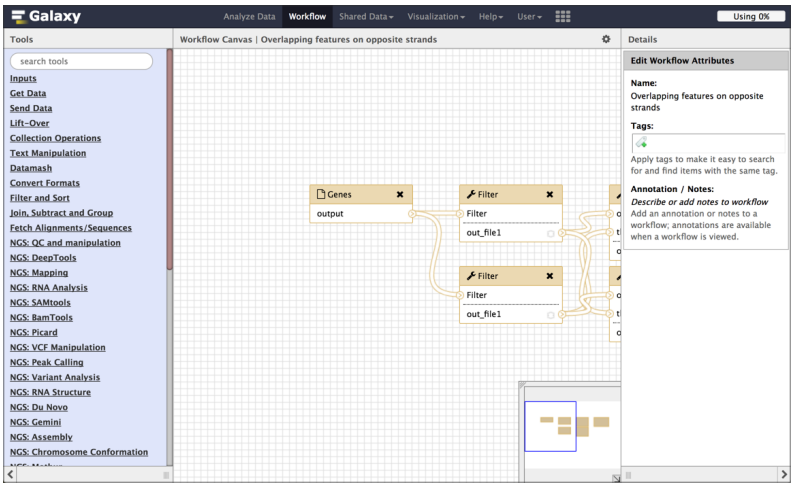

Figure 3.8 – Capture d'écran de l'éditeur de workflows Galaxy issue d'un tutoriel du site du projet Galaxy[77].

3.4.2 . Test de workflows

La représentation abstraite des pipelines fournie par les systèmes de workflows facilite la mise en place de tests unitaires qui se concentrent sur des étapes du workflow.D'autres tests sont les tests d'intégration, qui permettent de s'assurer que les différentes parties du workflow s'assemblent correctement et fonctionnent ensemble de manière cohérente, et les tests de déploiement, qui vérifient que que le workflow peut être exécuté avec succès dans l'environnement cible.
Snakemake propose une fonctionnalité dédiée aux tests unitaires [30]. Elle génère

30. https://snakemake.readthedocs.io/en/stable/snakefiles/testing.html

automatiquement des tests unitaires à partir d'un workflow (sous la condition que le workflow soit fonctionnel et se soit déjà exécuté d'une façon satisfaisante pour l'utilisateur) et d'un jeu de données test, lors d'une exécution.
Ces tests peuvent être conçus pour la vérification des sorties générées ou la détection d'erreurs potentielles. Ils peuvent ensuite être exécutés de manière automatisée, ce qui permet de détecter rapidement tout problème éventuel dans le workflow, garantissant ainsi sa fiabilité et sa qualité.
De façon complémentaire, les utilisateurs peuvent spécifier des règles test dans leurs workflows Snakemake, notamment avec Pytest [78]. Par exemple, une règle de test peut spécifier les fichiers de référence et les conditions qui doivent être satisfaites pour que le test réussisse. Des messages d'erreurs détaillés sont ensuite générés par Snakemake pour décrire les erreurs rencontrées.
Une infrastructure de tests pour Nextflow est l'outil *nf-test* [31]. nf-test permet d'implémenter et d'exécuter des tests unitaires pour chacun des processeurs, ou pour le workflow entier. Contrairement à Snakemake, les tests unitaires ne sont pas générés automatiquement.

3.4.3 . Déploiement

Les workflows orchestrent toutes les couches d'analyse décrites dans la figure 3.1 (traitement, exécution, outils et environnement).
Ils planifient l'exécution des tâches, surveillent le bon déroulement de leur exécution, gèrent leur soumission et re-soumission en cas d'échec, sur une grande diversité d'infrastructures HPC avec un effort minimal de la part du développeur.
Plus précisément, les systèmes de workflows permettent de complètement dissocier les machines d'exécution sous-jacentes (locales, *clusters*, *cloud*, etc.) de la mise en œuvre du workflow, en séparant la logique du workflow de sa configuration (quelle machine et quel ordonnanceur il utilise). En ce qui concerne l'optimisation de l'exécution, les systèmes de workflows ajoutent une couche de planification des tâches au-dessus du système d'exploitation, qui exécute la tâche, et de l'ordonnanceur HPC (*e.g.*, SLURM). Comme les workflows permettent d'exécuter des tâches indépendantes en parallèle, la couche de planification des tâches permet de distribuer les tâches sur toutes les ressources de calcul disponibles, tout en s'assurant qu'elles s'exécutent dans le bon ordre. Snakemake et Nextflow font partie des systèmes de workflows prenant en charge la planification distribuée [79].
Le paramétrage de l'utilisateur se limite à la spécification de l'infrastructure et des ressources matérielles disponibles. Pour Snakemake et pour Nextflow, par exemple, les caractéristiques de l'infrastructure sont à spécifier dans un fichier de configuration sous le format JSON.

31. https://code.askimed.com/nf-test/

Un fichier de configuration des paramètres d'un cluster (avec comme ordonnanceur SLURM) pour Snakemake ressemblerait à :

```
{
    "__hpc_config__":
    {
        "account": "compte_marine",
        "time": "01:00:00",   # Temps maximal d'exécution par tâche
        "cpus-per-task": 8,   # Nombre de cœurs alloués par tâche
        "mem-per-cpu": "4G",   # Mémoire allouée par cœur
        "partition": "partition_marine",    # Nom de la partition du cluster
        "output": "logs/slurm-%j.out",    # Fichier de sortie pour les sorties
        "error": "logs/slurm-%j.err"      # Fichier de sortie pour les erreurs
    }
}
```

Le même fichier sous Nextflow serait :

```
process {
    executor = 'slurm'
    queue = "queue_marine"
    time = '01:00:00'   // Temps maximal d'exécution par tâche
    cpus = 8         // Nombre de cœurs alloués par tâche
    memory = '4G'   // Mémoire allouée par tâche
    // Paramètres pour la sortie des tâches
    errorStrategy = 'finish'
    errorExitStatus = 2
}
// Configuration pour la sortie des tâches et des erreurs
report {
    enabled = true
    outputs = file('logs/slurm_out')
    errors = file('logs/slurm_err')
}
```

3.4.4 . Maintenance et reproductibilité

Les pipelines bioinformatiques font face à plusieurs défis, notamment la gestion des dépendances logicielles et la reproductibilité des analyses. Nous avons présenté dans la section 3.3.5 des solutions d'encapsulation des dépendances logicielles : les hyperviseurs, les conteneurs et les systèmes de gestion de paquets, qui facilitent

l'installation, la configuration et l'exécution des outils bioinformatiques.
Ces outils sont un grand pas en avant pour la maintenance, la reproductibilité et la réutilisation. Cependant, il n'est pas toujours trivial de les intégrer dans les pipelines, cette étape relevant de la responsabilité du développeur. Encore ici, les systèmes de gestion de workflows sont d'une grande aide pour orchestrer certaines de ces solutions avec les pipelines.

Certains d'entre eux, notamment Snakemake et Nextflow, permettent d'associer un conteneur (Docker/Singularity) ou un environnement (Conda) à chaque processeur via les paramètres du workflow. Ce faisant, ils dissocient la mise en œuvre de chaque étape de sa configuration d'environnement (qui détermine le conteneur dans lequel elle s'exécute).

Cette fonctionnalité, couplée à la nature modulaire des workflows, est un atout majeur qui facilite leur maintenance. Si un utilisateur souhaite modifier une étape, par exemple en changeant la version d'un outil, il lui suffit de modifier le processeur correspondant et son environnement associé. L'utilisateur ne doit se soucier que de la cohérence des inputs et outputs du processeur, et cela a un impact limité sur le reste de son code.

3.4.5 . Partage et réutilisation

Les systèmes de workflows scientifiques permettent également une meilleure réutilisabilité et partageabilité des workflows. Deux éléments de solution existent : l'encapsulation et la modularité des workflows d'une part, et d'autre part la possibilité de rechercher des workflows dans des plateformes de partage de workflows.

Modularité des workflows

Les processeurs, éléments constitutifs des workflows, peuvent être isolés et réutilisés pour former de nouveaux workflows. Un groupe de processeurs peut être encapsulé dans un sous-workflow, qui peut lui aussi être réutilisé pour former de nouveaux workflows.
L'encapsulation et la modularité facilitent la réutilisation des unités individuelles des workflows et cachent partiellement leur complexité [11].
En outre, les langages des systèmes de gestion de workflows permettent d'implémenter des pipelines dans un cadre standardisé ce qui peut faciliter leur lecture par d'autres utilisateurs.

Snakemake propose plusieurs niveaux d'encapsulation et de modularité.
D'une part, Snakemake permet l'inclusion de processeurs provenant de sources ex-

ternes. La fonctionnalité *module* permet de définir un workflow externe comme un "module", et d'en importer un ou plusieurs processeurs (ou des fonctions définies dans ce workflow). Il est aussi possible d'encapsuler un processeur (et un environnement Conda) dans un *wrapper*. Ils peuvent être inclus dans des workflows grâce à la directive *wrapper*. Cette fonctionnalité permet aux utilisateurs de se constituer une bibliothèque de *wrappers* facilement réutilisables. D'autre part, Snakemake permet l'inclusion de workflows via la fonctionnalité *include*.
Nextflow offre des fonctionnalités d'encapsulation et de modularité accrues depuis la deuxième version de son langage (DSL2) en 2020.
Nextflow permet d'encapsuler dans des *modules* des fonctions, des processeurs, mais aussi des sous-workflows. Ces modules peuvent être inclus dans de nouveaux workflows grâce à la directive "include". Il est possible d'inclure plusieurs modules à la fois.
Snakemake et Nextflow permettent d'inclure un workflow écrit dans un autre langage, ce qui favorise leur interopérabilité. Cela peut permettre aux utilisateurs de réutiliser un workflow tel quel, sans avoir à le réécrire.

Note sur l'interopérabilité des workflows
Il existe un standard ouvert de représentation des workflows scientifiques, le CWL [80]. Il a l'avantage d'être interopérable et portable, pouvant être utilisé sur n'importe quelle plateforme prenant en charge Docker ou Singularity (au même titre que Nextflow). Une différence majeure entre CWL et Nextflow ou Snakemake, est que le CWL est un langage et pas un moteur d'exécution, contrairement à Nextflow et Snakemake. CWL définit un format de description des workflows qui décrit les étapes, les entrées, les sorties et les dépendances des tâches dans un workflow. Cela permet d'avoir une représentation générique et portable des workflows, indépendamment du langage de programmation utilisé pour implémenter les tâches individuelles.
Par ailleurs, CWL utilise des formats standard, YAML et JSON, pour décrire les workflows, tandis que Nextflow et Snakemake utilisent leurs propres Domain Specific Language (DSL).
On peut noter que l'utilisation de YAML dans CWL le rend plus facilement lisible par des non spécialistes, tandis que l'utilisation des langages spécifiques de Nextflow et Snakemake nécessite un peu plus de connaissances préalables. Cependant, les langages utilisés par Nextflow et Snakemake sont des langages de programmation impératifs, ce qui leur confère une plus grande flexibilité que CWL en leur permettant de gérer des cas complexes et des flux de données dynamiques, là où CWL définit les dépendances de façon statique.

Plateformes de partage de workflows

Nous pouvons distinguer trois types de plateformes de partage de workflows [81] : **i**) celles qui sont gérées par les développeurs de systèmes de gestion de workflows (WfMS) (seuls les développeurs peuvent téléverser les workflows), **ii**) les plateformes de partage de workflows gérées par la communauté (les workflows sont sélectionnés par la communauté) et **iii**) les collections de workflows non gérées (tout le monde peut participer et les workflows ne sont pas examinés lors de leur ajout à la plateforme).
Dans la catégorie **i**), nous avons *snakePipes*, une plateforme qui contient une dizaine de workflows Snakemake.
Dans la catégorie **ii**), il y a, pour Snakemake *sequana* [82], contenant une douzaine de workflows et *nf-core* pour les workflows Nextflow [76], qui en contient 85 (en Juillet 2023).
Dans la catégorie **iii**), nous avons la *base de données de workflows Galaxy*[32] sur le serveur public le plus important de Galaxy, qui en contient une centaine, ainsi que *Snakemake workflow catalog*[33], qui en contient 2 269 (issus de GitHub). *WorkflowHub* [83], une plateforme plus généraliste, contient une centaine de workflows de plusieurs langages de workflows.
D'autres plateformes de partage de workflows ont été développées dans le passé, telles que myExperiment [84], CrowdLabs [85] ou SHIWA [86] pour n'en citer que quelques-unes. Ces dépôts ne sont plus maintenus car ils étaient associés aux systèmes Taverna, Kepler et VisTrails qui ne sont aujourd'hui plus maintenus.

On peut noter que GitHub est la source la plus importante de workflows, avec plusieurs milliers de workflows de bioinformatique disponibles. Nous y reviendrons dans le prochain chapitre.

Métadonnées et fonction dans les plateformes de partage

Il est important, pour faciliter l'indexation et la recherche des workflows, de leur associer des métadonnées décrivant précisément leur fonction et leurs caractéristiques [87].
On peut stocker de telles informations sous forme de descriptions ou de *tags*.
Sur toutes les plateformes de partage que nous avons évoqué, il est possible pour les utilisateurs d'annoter leurs workflows avec des *tags* issus d'une folksonomie. Une folksonomie est un système de classification collaboratif, non hiérarchisé et dont les termes sont proposés par un ensemble d'utilisateurs [88].

32. https://usegalaxy.org/workflows/list_published
33. https://snakemake.github.io/snakemake-workflow-catalog

Certaines plateformes, comme *Galaxy* et *Workflow-hub* proposent l'annotation des workflows avec un vocabulaire contrôlé, l'ontologie EDAM. L'utilisation d'une ontologie permet d'exploiter sémantiquement les annotations : comme les concepts sont liés dans un cadre formel et structuré, il est possible de mesurer une distance sémantique entre eux, qui est plus précise que des distances calculées dans des folksonomies.
Il est important que les annotations soient le plus représentatives possible de la fonction des workflows. Or, sur les plateformes existantes, ce sont les utilisateurs qui doivent les renseigner manuellement. Leurs qualité et quantité sont donc souvent insuffisantes [87].

Les annotations utilisées pour décrire les workflows peuvent varier d'une plateforme à l'autre. Si l'on peut retrouver certaines d'entre elles (nom du workflow, nom de l'auteur,...) dans tous les dépôts, on ne peut pas établir de correspondance pour d'autres (il est possible de préciser la liste d'outils dans WorkflowHub mais pas dans les autres plateformes, une annotation unique à Snakemake workflow catalog est la date du dernier commit du dépôt GitHub du workflow,...). Le développement de normes de métadonnées standardisées pour décrire les workflows aiderait à les rendre trouvables et favoriserait l'interopérabilité entre les plateformes de workflows.
L'automatisation de l'annotation des workflows peut permettre de pallier ce problème et d'obtenir plus d'annotations.

3.4.6 . Exemple de workflows

Le projet ICAN est une très bonne illustration des besoins rencontrés dans les projets de santé multidisciplinaires à grande échelle. En particulier dans de tels projets, en raison des contraintes légales liées à la protection des données personnelles, il n'est pas possible de déplacer des données d'un site partenaire à un autre. Les pipelines doivent être déployés sur le site du partenaire détenteur des données. Il est donc crucial de suivre de bonnes pratiques pendant tout le cycle de vie du pipeline. Cela facilite leur partage entre les partenaires, garantissant qu'il fonctionne correctement et fournit les résultats attendus, même dans des environnements HPC différents des environnements de développement et de test.
L'utilisation de standards ouverts plutôt que de langages propriétaires pour spécifier le pipeline et sa configuration, ainsi que les environnements logiciels et d'exécution sont importants pour obtenir un bon niveau de reproductibilité.

Nous introduisons un exemple de workflows développé dans le contexte du projet ICAN qui effectue l'analyse des données (contrôle qualité). Notre workflow a

été développé dans Snakemake, il s'appelle *BAM_QC*[34] et est représenté dans la figure 3.9. Ce workflow évalue la qualité des données de séquençage brut en calculant plusieurs métriques (*e.g.* longueur de séquence, niveau de contamination ou pourcentage de séquences alignées sur le génome de référence) sur tous les échantillons. Les résultats de ce workflow sont utilisés pour décider d'accepter ou de rejeter les échantillons.

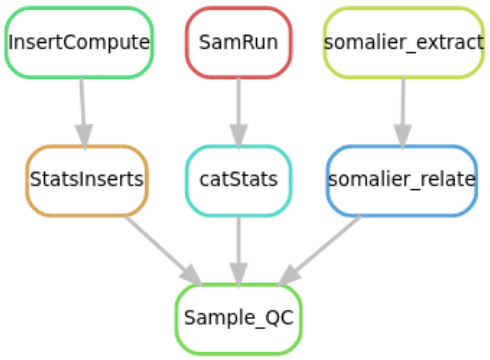

Figure 3.9 – Représentation schématique de BAM_QC, un workflow de contrôle qualité d'échantillons. Il est composé de 7 étapes : *i*) InsertCompute et StatsInserts estiment la longueur des fragments des séquences, **ii**) SamRun et catStats calculent plusieurs statistiques sur le mapping, et **iii**) somalier_extract et somalier_relate calculent la parenté entre les échantillons.

Chaque étape du workflow est mise en œuvre sous la forme d'une *règle* dans Snakemake (voir Figure 3.10). Une règle spécifie i) ses entrées et sorties, ii) le script à exécuter, et iii) sa configuration (environnement logiciel et machines d'exécution à utiliser), généralement documentée dans un fichier indépendant. Bien que ce workflow soit assez simple, il aurait été plus difficile de satisfaire toutes les exigences décrites dans la section 2 (en termes de réutilisation, de maintenance, etc.) sans un système de workflows scientifique.
En utilisant un système de workflows, le code du pipeline (qui implémente la logique de l'expérience ou de l'analyse de données) est découplé des ensembles de données d'entrée et de la configuration. Cela permet au même workflow d'être portable et exécutable sur différentes plateformes/environnments, et sur différents ensembles de données d'entrée.

3.5 . Conclusion

Le développement de pipelines complexes d'analyse de données, qui soient reproductibles et réutilisables, présente de nombreux challenges.

34. Code disponible sur https://github.com/r-blanchet/BAM_QC

```
rule InsertCompute:
  input:
    cram = "sample1.cram","sample2.cram"
  output:
    pdf = temp("{sample}.insert_size_histogram.pdf"),
    txt = temp("{sample}.insert_size_metrics.txt"),
  params:
    reference = "reference_genome.fasta",
  threads: 1
  Conda: "insert_env.yaml"
  shell:
    """
    SAMPLE='samtools samples {input.cram}|cut -f1'
    picard CollectInsertSizeMetrics -I {input.cram} -O
    ${{SAMPLE}}.insert_size_metrics.txt -H
    ${{SAMPLE}}.insert_size_histogram.pdf -M 0.5 -R {params.reference}
    """
```

Figure 3.10 – Exemple d'une règle Snakemake qui estime la longueur des fragments séquencés. Elle spécifie ses entrées (deux fichiers cram), ses sorties (un fichier pdf et un fichier txt), ses paramètres (un fichier fasta de référence), son environnement (Conda, défini dans un fichier YAML associé), le nombre de threads qu'elle va utiliser, ainsi que le script à exécuter (impliquant l'appel à deux outils de bioinformatique, à savoir *Samtools* et *Picard* [26]).

De nombreuses solutions existent pour implémenter des pipelines capables d'orchestrer des scripts provenant de plusieurs langages de programmation et plusieurs outils. Les systèmes de gestion de workflows sont la solution la plus adaptée à ce problème, en permettant aux utilisateurs de concevoir et d'exécuter des pipelines sous forme de modules reliés par des flux de données. Snakemake et Nextflow font partie des systèmes actuels les plus populaires et offrent aux utilisateurs un contrôle important sur la conception, la configuration et l'exécution des workflows. Ces systèmes de workflows répondent à bon nombre des défis auxquels sont confrontés les développeurs et les utilisateurs de pipelines.
La conception des pipelines est facilitée par le registre bio.tools qui catalogue des outils bioinformatiques et en propose une description standardisée et structurée basée sur des annotations sémantiques. Ce registre répond aux besoins des bioinformaticiens de trouver facilement et comparer des outils (**CU1**).
Le test des pipelines, étape importante pour s'assurer de leur bon fonctionnement en phase de développement aussi bien qu'en phase finale, nécessite l'écriture de scripts de tests par l'utilisateur. Nextflow et Snakemake offrent un cadre de tests de pipelines, et Snakemake propose des fonctionnalités de génération et d'exécution automatique de tests unitaires. Ces fonctionnalités aident les bioinformaticiens souhaitant s'assurer de la fiabilité de leurs pipelines (**CU6**) à mettre en place des tests. Cependant, la création de jeux de données de tests reste une tâche qui incombe à l'utilisateur.

Les environnements de calcul HPC permettent aux utilisateurs d'exécuter des pipelines coûteux en temps et en ressources (**CU7**). L'exécution de pipelines sur des environnements HPC est facilitée par Nextflow et Snakemake, qui automatisent l'optimisation de l'exécution des pipelines et la communication avec les ordonnanceurs de tâches.
La description et le partage des environnements logiciels des pipelines est une condition importante pour les rendre reproductibles. Trois catégories principales de solutions d'encapsulation des dépendances logicielles existent. D'un côté, les hyperviseurs fournissent une encapsulation "lourde" très complète mais sont peu partageables. D'autre part, les conteneurs et les systèmes de gestion de paquets proposent des encapsulations plus légères mais sont axés sur le partage. Leur réutilisation est favorisée par des plateformes de partage, dont l'initiative *Biocontainers*, spécifique à la bioinformatique. Nextflow et Snakemake offrent des fonctionnalités d'association de conteneurs Docker et Singularity et d'environnement Conda aux processeurs. Ces fonctionnalités tirent parti de la nature modulaire des workflows, facilite leur maintenance **CU2** et la réutilisation des processeurs.
La génération d'une trace d'exécution permet aux utilisateurs de comprendre comment des données ont été générées **CU4** et de les rendre reproductibles et donc fiables. De nombreuses ontologies permettent de représenter la provenance dans un format structuré. Snakemake et Nextflow ne génèrent que des données de provenance limitées et Galaxy propose des informations plus fournies, mais aucun de ces trois systèmes ne spécifie les traces de provenance sous les standards établis par la communauté.
Les systèmes de gestion de workflows facilitent la création de workflows reproductibles. Ils intègrent également de nombreuses fonctionnalités pour faciliter la réutilisation, notamment les fonctionnalités d'inclusion de workflows et de processeurs de Snakemake et Nextflow.

Si ces fonctionnalités facilitent la réutilisation, il existe de nombreux obstacles à son application "réelle". Premièrement, les workflows doivent être **disponibles**. De nombreuses plateformes de partage de workflows existent, mais seul un faible nombre de workflows sont disponibles sur ces plateformes.
Deuxièmement, les workflows doivent être **trouvables**. Les utilisateurs veulent pouvoir les trouver au sein d'une base de données **CU8**. Pour ce faire, il est important qu'ils soient annotés avec des informations précises sur leur fonction et leur contenu pour faciliter leur recherche. Si dans les bases de workflows actuelles, des moteurs de recherche sont proposés, les annotations elles sont dépendantes des utilisateurs, et donc de qualité très variable. Par ailleurs, ces annotations sont

hétérogènes entre les plateformes.

Des bases de données que nous avons évoquées, comme WorkflowHub et myExperiment proposent des fonctionnalités de citation de pipelines **CU9**. Il est possible pour les auteurs de préciser quels workflows de la plateforme ils ont réutilisé dans les annotations liées à leurs workflows. Cette fonctionnalité offre la possibilité aux auteurs de workflows non publiés dans des journaux d'être cités, mais elle est limitée : on ne peut citer que des workflows de la même plateforme. Par ailleurs, cette fonctionnalité ne permet pas de savoir quels fragments du pipeline ont été réutilisés, ni quelle version du workflow est concernée par la réutilisation.
La plupart des problèmes soulevés dans les cas d'utilisation sont couverts, au moins partiellement, par un écosystème de solutions techniques, en particulier les systèmes de gestion de workflows. Cependant, les cas d'utilisation **CU1** (complétion de pipelines) et **CU5** (adaptation de pipelines aux données), relatifs à la réutilisation, sont des questions ouvertes.
Ils soulignent des limites qui subsistent pour mettre en place des mesures favorisant la réutilisation.
Nous ouvrons cette section sur deux problèmes liés à deux catégories de réutilisation.
Premièrement, pour pouvoir favoriser la réutilisation de code de workflows, notamment en permettant aux utilisateurs de compléter leurs pipelines, il est avant tout nécessaire de pouvoir les comparer, notamment de mesurer leur similarité. Si on peut mesurer la similarité entre deux workflows, un utilisateur qui cherche à compléter son workflow peut par exemple rechercher dans une base de workflows une liste de candidats similaires dont il pourrait s'inspirer.
Deuxièmement, pour favoriser la réadaptation de workflows à de nouvelles données, il est nécessaire de guider l'utilisateur dans le choix des éléments du workflow à modifier. Le nombre de paramètres d'un workflow peut vite devenir important, augmentant d'autant plus le nombre de combinaisons possibles parmi lesquelles il sera difficile de faire un choix.

Problème ouvert : Recherche de similarité dans les workflows

La recherche de similarité entre workflows présente plusieurs intérêts. En parvenant à la définir, il devient possible de capitaliser sur les connaissances et les bonnes pratiques déjà établies pour réutiliser des workflows en s'inspirant de modèles existants, et en les adaptant au lieu de les concevoir à partir de zéro. On peut également identifier des standards et pratiques courantes en établissant des liens de similitudes entre les workflows. Ce faisant, on peut identifier les motifs ou combinaisons courantes, ou encore harmoniser les workflows.

Il existe deux familles de solutions pour la mesure de similarité des workflows : celles basées sur l'analyse de la structure du workflow, et celles basées sur les métadonnées du workflow [89].
L'analyse par la structure comporte deux catégories de méthodes, la détection de motifs d'une part et la comparaison de graphes d'autre part. La détection de motifs cherche à identifier des sous-workflows fréquemment présents au sein des workflows, tandis que la comparaison de graphes quantifie les distances entre des graphes.
D'autres méthodes de mesure de similarité sont axées sur l'analyse des métadonnées associées aux workflows, comme les types de données utilisés, les paramètres, les entrées et sorties, les annotations et les propriétés fonctionnelles. Les métriques de similarité basées sur les métadonnées peuvent être utiles dans le cas où l'on compare des workflows qui effectuent des tâches similaires, mais qui ont des structures différentes.
Des travaux précoces sur la recherche de similarité dans les workflow [90] ont proposé des méthodes de *découverte* de workflows.
Ils distinguent deux cas différents dans lesquels les utilisateurs cherchent à découvrir des workflows : lorsqu'ils cherchent à réutiliser, et lorsqu'ils cherchent à réadapter. Alors que la réutilisation nécessite de trouver des workflows pertinents relativement à une requête donnée, la réadaptation nécessite de trouver des workflows similaires ou complémentaires à un workflow donné.
Les auteurs ont implémenté et comparé les deux familles de solutions via deux méthodes de recherche de workflows similaires à un workflow donné wA parmi un ensemble E de workflows Taverna.
La première méthode [91], basée sur la structure, compare wA à E avec un outil de comparaison de graphes [92]. Les résultats de cet outil sont ensuite utilisés pour classer les workflows de E du plus similaire au moins similaire à wA.
La deuxième méthode, basée sur les annotations, prend en compte le code des workflows, plus précisément les services web auxquels ils font appel. Cette méthode convertit les workflows de E en "sacs de services web", et les regroupe dans des *clusters* via un algorithme de clustering de texte [93]. Ces clusters sont ensuite utilisés pour générer, pour wA une liste ordonnée de workflows similaires provenant de E.
Les auteurs constatent que la méthode basée sur la structure tend à détecter la similarité de manière plus précise dans le cas de comparaison de versions différentes d'un même workflow, alors que celle basée sur les métadonnées détectent mieux la similarité inter-workflows.
Cependant, l'étude évalue les résultats des méthodes comme insatisfaisants. Une

des limites de ces méthodes est qu'elle ne permettent de prendre en compte que des informations limitées pour la comparaison des workflows. La prise en compte d'annotations supplémentaires améliorerait la détection de workflows similaires.
Plus tard, une étude comparative d'algorithmes de recherche de similarité pour les workflows [89] a réimplémenté, comparé et classifié non seulement ces algorithmes mais aussi des combinaisons de ceux-ci. Cette revue constate que la recherche de similarité basée sur les annotations donne les meilleurs résultats, en accord avec les études précédentes. Cependant, elle indique qu'au vu de la variabilité de la qualité et de la quantité des annotations, cela n'est pas une méthode fiable pour comparer les workflows dans des cas d'utilisation réels. Ils constatent également que la recherche de similarité basée sur la structure peut donner des résultats presque aussi bons que ceux basés sur l'annotation. Selon la nature des workflows comparés, la combinaison de plusieurs algorithmes basés sur la structure peut également être meilleure qu'un algorithme individuel.
De façon similaire aux travaux précédents, cette étude observe que, si les méthodes basées sur les annotations sont les plus performantes, elles sont difficilement applicables dans des bases de données de workflows réelles, car les workflows ne sont pas tous correctement annotés. Ils soulignent que l'apport d'informations contextuelles sur les workflows et leurs modules (comme connaître le type d'un module, ou si un module est fréquemment retrouvé dans un ensemble de workflows) peut réduire la complexité computationnelle des algorithmes de recherche de similarité et améliorer la qualité des résultats, mais l'acquisition de telles connaissances n'est pas trivial à automatiser.

Problème ouvert : Recommandation de paramètres d'un workflow

Réutiliser un workflow en l'exécutant sur des données différentes de celles pour lequel il a été créé présente de nombreux challenges. Au delà d'être capable de lire le workflow, une compréhension très précise de chacune des étapes est requise. La documentation est utile pour comprendre quels paramètres ajuster, mais elle peut être incomplète et manquer d'informations pour que l'utilisateur s'approprie le workflow.
Pour pallier ce problème, une méthode basée sur l'apprentissage automatique pour suggérer des paramètres adaptés à de nouveaux jeux de donnée, FReeP, a été proposée [94]. À partir d'un ensemble de paramètres initiaux entrés par l'utilisateur, le système propose un ou plusieurs paramètres pour les compléter. FReeP déduit des paramètres en se basant sur des traces de provenance du workflow (spécifiées dans un des formats standard), qui décrivent des exécutions passées. Cet algorithme donne des résultats prometteurs et son intérêt réside dans la possibilité de l'appliquer à un grand nombre de systèmes de gestion

de workflows. Cependant, ses performances sont liées à la quantité de données de provenances auxquelles il a accès. La plateforme de partage de données de provenance, ProvStore [95] [35], contient 3 662 documents (en Juillet 2023), mais la plupart des plateformes de partage de workflows populaires, telles que WorkflowHub, ne proposent pas de stocker les données de provenance. Les systèmes de gestion de workflows Snakemake et Nextflow ne génèrent pas automatiquement de données de provenance fournies. Galaxy en revanche, génère et stocke des historiques d'exécutions, mais ils ne sont pas sous un format standard, et ne sont pas toujours disponibles sur des dépôts publics.
D'autre part, l'application de FReeP dans des conditions réelles présente certaines limitations, notamment en termes de *scalabilité* : son temps d'exécution est plus long que l'exécution du workflow lui-même. De plus, l'algorithme ne tient pas compte de la pertinence des ensembles de paramètres.

35. https://openprovenance.org/store/

4 - Étude de la réutilisation

Sommaire

4.1 . Introduction

Dans ce chapitre, nous présentons la contribution majeure de ce travail de thèse : les résultats d'une étude que nous avons menée sur la réutilisation de workflows scientifiques. Notre objectif est double : montrer comment les systèmes de workflows sont utilisés dans la pratique par la communauté de la bioinformatique et comprendre les pratiques de réutilisation. Alors que les études précédentes (*e.g.*, [96]) se sont concentrées sur les systèmes de workflows Taverna et Galaxy, notre étude est la première à considérer les systèmes de workflows de plus en plus populaires que sont Nextflow et Snakemake.
Plus précisément, nous avons étudié la réutilisation de workflows Nextflow et Snakemake disponibles sur des dépôts publics GitHub.
Pour étudier la réutilisation, comme mis en évidence à la fin du chapitre précédent, une problématique centrale est celle de la similarité des workflows : Comment déterminer si deux workflows sont similaires ? Quels critères prendre en compte ? Comment mesurer cette similarité ? Ce chapitre décrit une démarche et une analyse de données ayant pour objectif la mesure de la similarité entre deux workflows bioinformatiques en Nextflow et Snakemake. La similarité entre deux workflows est ici définie comme une similarité "fonctionnelle". On cherchera à savoir si deux workflows ont la même fonction, c'est-à-dire s'ils effectuent les mêmes opérations. Comme présenté au chapitre précédent 3.5, deux approches principales sont envisageables pour rendre compte de la similarité de deux workflows. Si l'approche par

annotation donne de meilleurs résultats lorsqu'elles sont de bonne qualité, les métadonnées des workflows sont de qualité variable. Dans notre cas, GitHub propose des fonctionnalités de description du dépôt (*tags*, description courte) et certains utilisateurs documentent leur dépôt (présence de *readme*), mais ces sources sont souvent peu informatives sur la fonction des workflows et difficiles à exploiter.
L'approche structurelle est moins dépendante du respect des bonnes pratiques de l'auteur du workflow mais donne de moins bons résultats que l'approche par annotations. Cependant, dans notre cas, nous étudions des fichiers de spécification de workflows Nextflow et Snakemake, et n'avons pas un accès direct à la structure. Notre recherche est principalement axée sur la recherche de similarité basée sur les annotations. Pour surmonter le problème du manque d'informations, nous développons un outil capable d'automatiquement trouver des annotations à partir de la spécification d'un workflow, en mettant l'accent sur la recherche de ses outils.
En effet, les outils bioinformatiques utilisés par un workflow offrent de précieuses indications sur les opérations qu'il effectue, permettant ainsi de mieux comprendre sa fonction. Les processeurs des workflows Nextflow et Snakemake peuvent contenir des scripts bash ou des commandes *shell*, dans lesquels on peut identifier des appels aux outils lancés en ligne de commande.
La suite de ce chapitre est structurée de la façon suivante. La section 4.2 décrit la démarche que nous avons suivie pour collecter les workflows de notre analyse. Nous introduisons nos résultats à travers trois points principaux. Tout d'abord, en section 4.3, nous étudions la réutilisation de workflows du point de vue des propriétaires et contributeurs de dépôts de workflows. Ensuite, en section 4.4 les pratiques de réutilisation sont examinées en comparant le code des workflows (copier-coller, forks, etc.). Troisièmement, la réutilisation des étapes de workflows individuelles entre les utilisateurs est analysée en section 4.5. Bien que ces aspects soient principalement quantitatifs, nous passerons ensuite à une perspective qualitative sur la réutilisation en examinant les opérations de bioinformatique exécutées dans les workflows collectés en section 4.6. La section 4.7 fournit une vue globale de la base de données et la base de connaissances que nous avons constituée. Nous concluons le chapitre en 4.8.
Le code source pour la collecte de données et les analyses de réutilisation est disponible sur trois dépôts GitHub : l'extraction d'informations à partir de workflows Snakemake[1], l'extraction d'informations à partir de workflows Nextflow[2], et les expériences et figures des articles [3].

1. https://github.com/mdjaffardjy/Snakemake_workflow_analysis
2. https://github.com/mdjaffardjy/AnalyseDonneesNextflow
3. https://github.com/mdjaffardjy/Reuse_in_processes

4.2 . Collecte d'un ensemble de workflows

Pour constituer notre base de données d'études, nous avons fait le choix d'utiliser GitHub. En effet, les plateformes de partage dédiées au workflows contiennent peu de workflows Nextflow et Snakemake 3.4.5. À l'inverse, le partage de code sur GitHub est aujourd'hui une pratique courante chez les bioinformaticiens, et la plateforme contient un grand nombre de dépôts sur lesquels se trouvent des workflows Nextflow et Snakemake.
La Figure 4.1 illustre notre approche pour extraire et collecter des informations sur les workflows que nous avons étudié. Nous avons implémenté et exécuté les trois composants logiciels (*CrawlWF*, *ParsWF* et *WF2BT*) pour rechercher, extraire, analyser et annoter les workflows Nextflow et Snakemake sur GitHub.
Cette approche, initialement élaborée et implémentée pour Snakemake, a été adaptée pour Nextflow lors d'un stage.

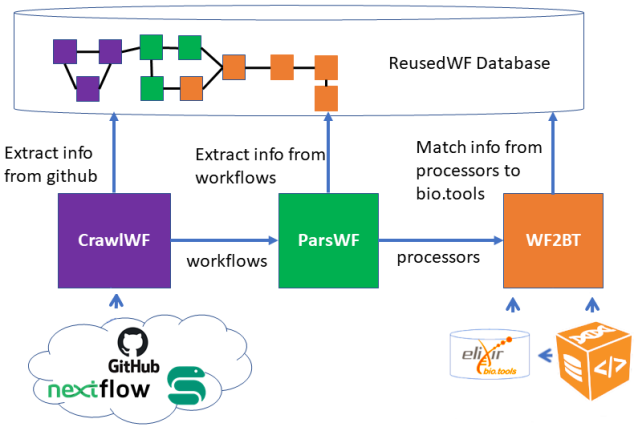

Figure 4.1 – Processus d'extraction et d'annotation des workflows. Il est composé de trois composants logiciels principaux : *i*) *CrawlWF* recherche et extrait les workflows depuis GitHub, *ii*) *ParsWF* analyse les workflows pour extraire des informations significatives, et *iii*) *WF2BT* associe les processeurs de workflows aux annotations de bio.tools.

CrawlWF collecte et extrait des workflows, Nextflow et Snakemake, depuis GitHub. Il recherche les dépôts contenant un fichier de spécification de workflow (.nf pour Nextflow, .smk ou Snakefile pour Snakemake). L'API GitHub a été utilisée pour retrouver ces fichiers, mais cette API présente des limitations que nous avons dû contourner pour retrouver le plus grand nombre de fichiers possibles. Premièrement l'API ne retourne qu'un maximum de 1000 résultats pour une requête donnée. Pour contourner ce problème, il nous faut donc segmenter la recherche. La recherche de fichier (qui nous permettrait de rechercher des fichiers par extension) ne permet pas de trier les résultats. En revanche, la recherche de dépôt

nous permet de trier les résultats, mais ne nous permet pas d'utiliser de wildcards ou certains caractères spéciaux (notamment les symboles . et *). Ce "type" de recherche ne permet d'explorer que les noms, les descriptions et les readme des dépôt. On recherche donc des dépôts contenant "nextflow" ou "snakemake" dans leur readme, en les triant par date de création, avec une requête par mois depuis la sortie de chacun des systèmes. Par exemple, on recherchera les dépôts dont le readme contient "snakemake" créés entre le 01/07/2014 et le 01/08/2014, puis du 01/08/2014 au 01/09/2014,... Dans chacun des dépôts trouvés, on recherche des fichiers de spécification de workflows tels que décrits précédemment. Cette démarche ne nous permet pas de trouver la totalité des fichiers de spécification publics de la plateforme mais nous permet de constituer une base conséquente de fichiers de spécification de workflows.
En parallèle, il récupère les métadonnées de GitHub des dépôts pour associer chaque workflow au propriétaire (qui possède le dépôt) et aux contributeurs (qui ont contribué au dépôt), mais également des informations comme les tags et description du dépôt, sa date de création, *etc.*
ParsWF vient ensuite examiner le contenu du dépôt et ne conserve que les fichiers de workflows bien formés en vérifiant leur syntaxe. Un workflow bien formé en Nextflow est un workflow qui est bien parenthésé.
En Snakemake, un workflow bien formé sera un workflow qui peut être compilé par le compilateur de Snakemake. ParsWF extrait ensuite des annotations sur le fichier et les (*processeurs*) : nom des processeurs, liste des entrées et sorties,...
ParsWF cherche également à reconstituer la structure des workflows. En Nextflow, il ne peut faire cette recherche que pour les workflows dont le dépôt ne contenait qu'un fichier (qu'il considère comme étant écrits dans la première version du DSL de Nextflow, DSL1). Pour Snakemake, il peut faire cette recherche sur tous les workflows, mais il utilise une fonctionnalité de Snakemake.
Enfin, *WF2BT* annote les étapes du workflow avec des métadonnées récupérées du registre bio.tools [97]. *WF2BT* procède en trois étapes : *i*) il extrait le script shell de l'implémentation du processeur ; *ii*) il extrait les noms de commandes du script shell pour constituer un ensemble d'outils candidats ; *iii*) il fait correspondre les outils candidats à la liste des noms d'outils dans bio.tools. Pour *ii*) établir la liste d'outils candidats, *WF2BT* cherche les noms de commande des outils dans les *shell*. Il commence par établir une liste de *mots clés* en examinant le premier mot de chaque ligne de commande (en prenant en compte les |, ;,...). Pour trouver les *candidats*, nous filtrons cette liste de mots clés grâce à des règles syntaxiques (en cherchant si le mot clé est un script, par exemple, ou si il est exempt de certains caractères spéciaux comme # ou =), mais aussi grâce à une liste d'exceptions. Cette liste d'exceptions, stockée dans un fichier .txt, est exhaustive

et écrite à la main. Elle est alimentée au fur et à mesure des explorations. Elle contient notamment une liste de commandes shell, ou bien des commandes comme "python" ou "Nextflow". Pour *iii*) éliminer les faux positifs de la liste de candidats, nous recherchons leur présence dans bio.tools. Nous recherchons la présence de chaque candidat dans un dump local au format xml de bio.tools[4] pour vérifier s'il existe une entrée correspondante. Cependant, pour certains outils, il n'y a pas de correspondance entre le nom de commande et le nom d'outil. Les outils candidats qui ne pouvaient pas être mis en correspondance avec une entrée dans le registre bio.tools (pas de correspondance avec un nom d'outil de bio.tools) ont été comparés aux métadonnées de biocontainers [73]. Chaque conteneur est livré avec une liste des commandes shell de l'outil, ainsi que son identifiant de bio.tools. De cette manière, *WF2BT* liste l'ensemble des outils utilisés par chaque processeur et qui ont une entrée dans bio.tools. Pour finir, *WF2BT* associe aux processeurs des annotations liées aux outils issues de bio.tools (et donc à la fonction du workflow) : les noms d'outils selon biotools et leur uri, les topics (sujets liés à tous les outils du workflow) et leur uri EDAM, des informations sur la fonction avec les opérations de l'outil (et leur uri EDAM), ainsi que les données d'entrée et de sortie (le type des données et leur format). Nous enrichissons également ces annotations, en recherchant des synonymes dans EDAM pour les opérations et les topics.

En utilisant la méthode décrite ci-dessus, *CrawlWF* a pu extraire un total de 1 790 workflows Nextflow et 3 866 workflows Snakemake jusqu'en mai 2022. La figure 4.2 illustre l'évolution du nombre de workflows Nextflow et Snakemake disponibles sur GitHub au cours des huit dernières années.

Pour constituer notre jeu de données de workflows, l'exécution de *CrawlWF* a nécessité 58 heures pour les workflows Snakemake et 42 heures pour les workflows Nextflow. Cet outil utilise l'API de GitHub pour rechercher les dépôts contenant des fichiers de spécification de workflows, et le nombre de requêtes permis par cet API étant limité, ce temps est incompressible.

Parmi ces workflows, *ParsWF* a conservé 1 675 workflows Nextflow bien formés et 2 946 workflows Snakemake bien formés.

Sur ces workflows, *WF2BT* n'a conservé que 1 186 workflows Nextflow et 1 257 workflows Snakemake *matched-tools* (contenant au moins un processeur associé à au moins un outil dans bio.tools), ce qui représente un total de 2 443 workflows retenus. Cette différence dans les pourcentages obtenus vient en partie de la différence de gestion des sous-workflows dans la spécification : il existe beaucoup plus de sous-workflows très courts en Snakemake, dont le rôle est d'être le liant entre les sous-workflows contenant des outils. Ces workflows contiennent respectivement 9

4. https://github.com/bio-tools/biotoolsschema

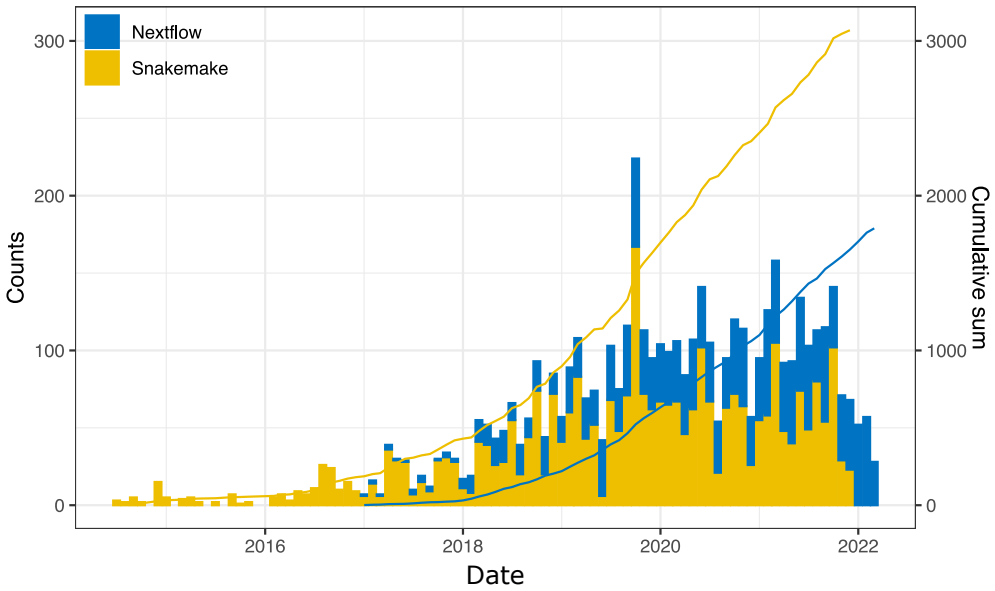

Figure 4.2 – Évolution mensuelle du nombre de workflows Nextflow et Snakemake dans GitHub depuis 2014. Les barres, superposées, représentent la quantité de workflows par mois pour chacun des systèmes de workflows et les lignes représentent la quantité de workflows au total pour chacun des systèmes.

652 et 5 888 processeurs *matched-tools* (des processeurs individuels pour lesquels au moins un outil a une entrée bio.tools).
Les **processeurs *matched-tools*** et les **workflows *matched-tools*** sont les objets d'intérêt dans le reste de l'étude.

4.3 . Propriétaires et contributeurs des workflows

Nous nous concentrons d'abord sur la répartition des workflows parmi les propriétaires de dépôts et les contributeurs. Pour ce faire, nous avons extrait ces informations de GitHub en utilisant *CrawlWF*. Chaque workflow est associé à un seul propriétaire (le propriétaire du dépôt GitHub) et éventuellement à plusieurs contributeurs (les utilisateurs GitHub ayant contribué au moins une fois au dépôt). Nous avons trouvé que les 1 186 workflows Nextflow et les 1 257 workflows Snakemake *matched-tools* étaient détenus respectivement par 650 et 535 propriétaires. Cela représente un total de 1 166 propriétaires de workflows (19 propriétaires ont publié à la fois des workflows Nextflow et Snakemake).
En ce qui concerne les dix propriétaires ayant fourni le plus grand nombre de workflows en Nextflow et Snakemake (les dix propriétaires qui possèdent le plus de workflows), ils ont en fait publié 15% de tous les workflows Nextflow et Snakemake.

Nous ne sommes donc pas dans une situation où les workflows sont partagés par un nombre restreint de propriétaires. Il est intéressant de noter que 31% (Snakemake) vs. 42% (Nextflow) des auteurs ont publié au moins deux workflows. Il existe donc un ensemble de propriétaires qui fournissent plusieurs workflows à la communauté, et dans l'ensemble, les auteurs de workflows produisent plusieurs workflows et sont disposés à partager leurs workflows.
Ces résultats ont été obtenus sur les propriétaires de workflows, mais des tendances similaires sont observées chez les contributeurs de workflows (dans Nextflow, 37% des contributeurs ont contribué à deux workflows ou plus, et ce chiffre atteint 44% pour Snakemake).

4.4 . Réutilisation dans les workflows

La question de la réutilisation des workflows est maintenant ici explorée en considérant la réutilisation complète (du workflow entier) plutôt que partielle (de parties du workflow).
Pour cela, deux métriques complémentaires ont été utilisées. Tout d'abord, nous avons recherché des correspondances exactes entre les textes des codes de workflows pour détecter les copier-coller entre les workflows. Ensuite, nous avons examiné le *forking* de projet de GitHub pour identifier la réutilisation des workflows. Le forking permet à un utilisateur Git de créer explicitement un nouveau projet à partir du code d'un projet parent, en copiant l'ensemble de son contenu et de son historique. Pour chaque projet de workflows, nous avons accès à son nombre de *forks*, et donc au nombre de fois qu'il a été utilisé comme source d'un nouveau workflow.
En conséquence, nous n'avons trouvé aucune paire de workflows avec un code source identique. Ensuite, en examinant le nombre de *forks*, nous avons obtenu des résultats intéressants. Le tableau 4.1 indique le nombre de workflows Nextflow et Snakemake *forkés* plus de n fois ($n = 3, 5, 10, 50$). Pour rappel, le nombre total de workflows est respectivement de $1,186$ et $1,257$ en Nextflow et Snakemake. Cela signifie, par exemple, qu'il y a 45 workflows Nextflow et 57 workflows Snakemake qui ont été *forkés* plus de 10 fois.

Threshold	3	5	10	50
Nextflow	13.7%	7.8%	3.8%	0.8%
Snakemake	14.5%	7.9%	4.6%	0.2%

Table 4.1 – Pourcentage de workflows Nextflow et Snakemake ayant été forkés plus de 3, 5, 10 et 50 fois.

Une observation des workflows issus de ces *forks* a révélé deux pratiques prédomi-

nantes : (i) un workflow est dupliqué, puis la copie subit des modifications pour le réorienter vers un nouveau contexte d'utilisation, (ii) un workflow populaire est dupliqué pour créer une version stable et inchangée pour ses utilisateurs actuels, permettant ainsi au workflow original de continuer à évoluer pour répondre aux besoins de nouveaux utilisateurs. Dans les deux cas, on considère que les workflows originaux et dupliqués ne sont pas identiques. On suppose donc que, bien que la réutilisation de workflows ait été identifiée par la présence de *forks*, les workflows (*forkés* ou originaux) ont été modifiés suffisamment pour ne pas être détectés comme identiques.

4.5 . Réutilisation des processeurs

Un autre objectif de cette étude est d'identifier la réutilisation à un niveau plus fin en détectant le copier-coller suivi de légères modifications de processeurs par les utilisateurs. La détection de similarités entre processeurs peut être effectuée à l'aide d'outils de détection de plagiat [98] ou en suivant les méthodes également utilisées dans d'autres études [9, 99]. Les outils de détection de plagiat sont plus adaptés aux codes plus longs qu'aux codes des *shell* des processeurs, nous avons donc choisi de suivre la même direction que les études précédentes sur la réutilisation [98] en utilisant la distance de Levenshtein (telle qu'implémentée dans jellyfish [5]), normalisée par la taille du processeur. On a donc d, distance de levenshtein normalisée, avec $levenshtein(p_1, p_2)$ étant la distance de levenshtein entre p_1 et p_2 et l_{max} le maximum entre la taille de p_1 et de p_2,

$$d(p_1, p_2) = (l_{max} - levenshtein(p_1, p_2))/l_{max}$$

En conséquence, la distance normalisée se situe entre 0 (aucune similarité) et 1 (codes identiques).
Pour compartimenter les processeurs en groupes de processeurs similaires, on considère initialement que l'on a un groupe par processeur. On va ensuite fusionner ces groupes en prenant comme condition la distance entre les processeurs : si elle est supérieure au seuil, alors les deux processeurs seront dans le même groupe. Ainsi, processeur par processeur, on constituera les groupes en regardant pour chaque processeur p les processeurs similaires p', et ensuite pour chaque p' les processeurs similaires p". Cela nous permet de mitiger le choix du processeur de départ en considérant une transitivité (avec le seuil considéré, on peut considérer que si p est similaire à p' et p' à p", p est similaire à p").
Pour identifier la réutilisation (sous la forme de copier-coller et légères modifica-

5. https://GitHub.com/jamesturk/jellyfish

tions), nous avons donc considéré que deux processeurs étaient similaires si leur score de similarité de Levenshtein était supérieur à $0,85$. Ce seuil a été choisi en fonction d'une inspection des processeurs similaires obtenus en considérant quatre valeurs de seuil ($0,80$, $0,85$, $0,90$ et $0,95$). En conséquence, 0,85 semblait être la meilleure valeur pour tenir compte de l'action de copier-coller suivie de légères modifications. Nous avons ensuite formé des "groupes" d'**occurrences de processeurs** en utilisant ce seuil. Chaque groupe représente un **processeur unique** regroupant toutes les occurrences de processeur très similaires.
La constitution des groupes **groupes_fusionnes** de processeurs similaires depuis la matrice de distance **M_dist** entre les processeurs **p_i** se fait en deux étapes :
i) on regroupe chaque processeur p_i dans un groupe groupe_x avec des processeurs p_j similaires à p_i ;
ii) on concatène ces groupes selon leurs éléments en commun.
L'algorithme de constitution des groupes est :

```
# Constitution des groupes
groupes <- []  # Liste pour stocker les groupes
p_parcouru <- []  # Liste pour stocker les indices parcourus

# Parcours de la matrice pour former des groupes "initiaux"
pour chaque ligne_p_i dans plage(0, longueur(M_dist)):
    # on ne parcourt pas les indices déjà dans des groupes
    si ligne_p_i n'est pas dans p_parcouru:
        groupe_x <- [ligne_p_i]
        ajouter ligne_p_i à p_parcouru
        # Itération sur les colonnes d'indice supérieur à "p_i"
        pour chaque colonne_p_j dans plage(ligne_p_i + 1, longueur(M_dist)):
            si M_dist[ligne_p_i][colonne_p_j] > 0.90 alors
                ajouter colonne_p_j à groupe_x
                ajouter colonne_p_j à p_parcouru
        ajouter groupe_x à groupes

# Concaténation des groupes contenant des éléments en commun

groupes_fusionnes <- []
pour chaque groupe_comparé dans groupes:
    intersectants <- [groupe_comparé]
    disjoints <- []
    intersectants <- []
    # si un groupe a des éléments en commun avec
```

```
    # un ou plusieurs groupes parcourus,
    # on les fusionne, sinon on l'ajoute à la liste
    pour chaque reference dans groupes_fusionnes:
        si groupe_compare intersecte reference:
            intersectants.ajouter(reference)
        sinon:
            disjoints.ajouter(reference)

    groupes_fusionnes = disjoints + [union(*intersectants)]

retourner groupes_fusionnes
```

On obtient via cet algorithme 5 375 processeurs réutilisés (ou groupes de processeurs) pour Nextflow et 3 239 processeurs réutilisés pour Snakemake.

Étude comparative de la réutilisation des processeurs entre Nextflow et Snakemake

Les histogrammes dans la Figure 4.3 représentent la distribution du nombre de workflows dans lesquels un processeur est réutilisé (réutilisation entre les workflows) dans Nextflow et Snakemake. Comme nous avons approximativement le même nombre de workflows pour Nextflow et Snakemake, nous ne normalisons pas la quantité de workflows.
L'histogramme de la Figure 4.3 montre que le profil de réutilisation est similaire entre Nextflow et Snakemake. Il est intéressant de noter que les processeurs les plus réutilisés de chacun des systèmes le sont plus fréquemment dans Nextflow que dans Snakemake. En regardant les "top-x processeurs" (les x processeurs qui sont trouvés dans le plus grand nombre de workflows), les processeurs Nextflow sont plus souvent réutilisés entre workflows : les 25 premiers processeurs sont réutilisés dans 3,34% (resp. 1,85%) des workflows Nextflow (resp. Snakemake) ; les 30 premiers processeurs sont réutilisés dans 2,42% des workflows Nextflow et 0,93% des workflows Snakemake.

Étude comparative de la réutilisation des processeurs Nextflow : rôle de nf-core

nf-core [76] étant un référentiel de workflows Nextflow bien connu, l'expérience suivante vise à enquêter sur l'impact de nf-core sur la réutilisation des processeurs dans les workflows Nextflow.
Dans GitHub, on peut identifier les workflows provenant de la plateforme nf-core comme ceux dont le propriétaire GitHub est "nf-core". Ainsi, dans cette expérience,

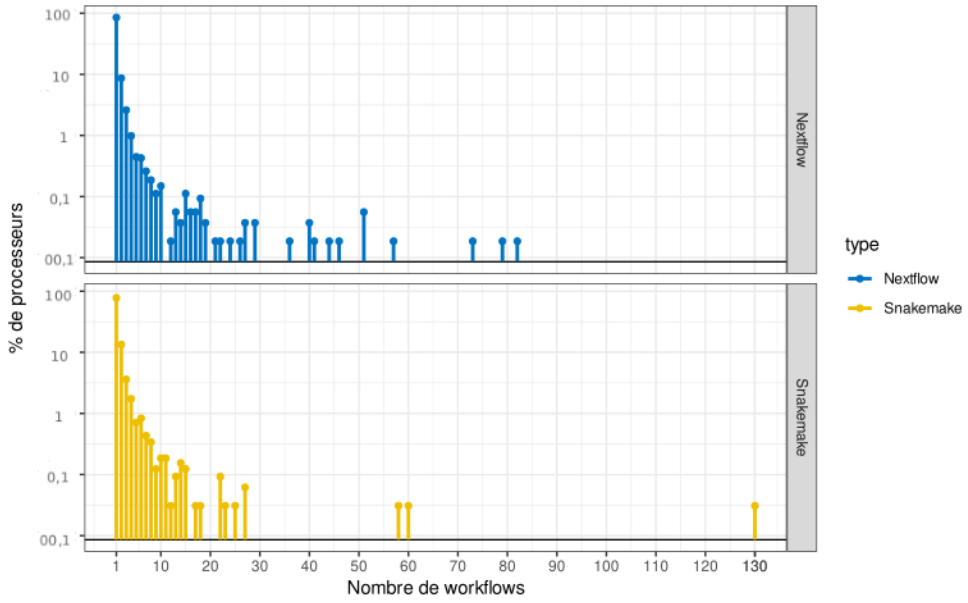

Figure 4.3 – Distribution de la réutilisation des processeurs dans les workflows Nextflow et Snakemake

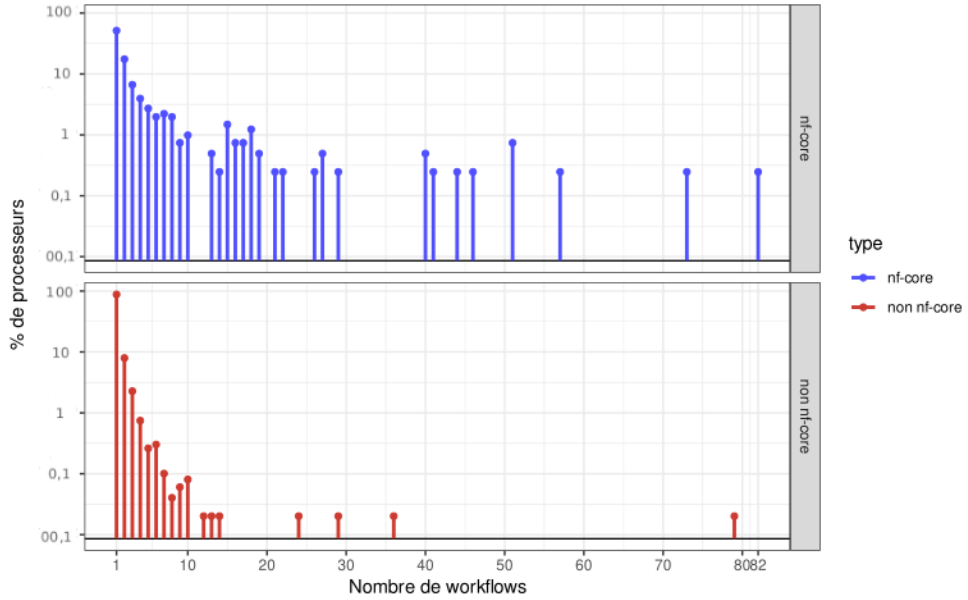

Figure 4.4 – Répartition de la réutilisation des processeurs dans les workflows non nf-core. Les processeurs nf-core et non nf-core sont pris en compte.

- un *workflow nf-core* est un workflow dont le propriétaire est nf-core,
- un *workflow non-nf-core* a un propriétaire différent de nf-core,
- un *processeur nf-core* est un processeur qui est utilisé dans au moins un

workflow nf-core,

- un *processeur non-nf-core* est un processeur qui n'est jamais utilisé dans un workflow nf-core.

En ne considérant que les workflows non-nf-core, la question est : y a-t-il une différence entre la réutilisation des processeurs nf-core et celle des processeurs non-nf-core ?
La figure 4.4 représente la distribution de la réutilisation des processeurs nf-core (en haut) par rapport à la réutilisation des processeurs non-nf-core (en bas) dans les workflows non-nf-core. On peut observer que la réutilisation des processeurs nf-core est légèrement plus élevée que la réutilisation des processeurs non-nf-core.

Note sur les wrappers et modules En Snakemake et en Nextflow, il existe des plateformes de partage de processeurs, conçus pour être facilement réutilisables par les développeurs.
En Nextflow, la plus importante plateforme est celle de nf core, qui met à disposition des *modules* nf-core. Ces modules, au nombre de 431, sont proposés et sélectionnés par la communauté nf-core, et disponibles sur la même plateforme que les workflows nf-core [6]. On peut y trouver un total de 431 modules, correspondant à 144 outils et leurs options, avec un module par outil+option (par exemple, samtools_index [7] et samtools_merge [8] sont deux modules distincts).
On retrouve 66 workflows dans lesquels sont réutilisés des modules nf-core.
Parmis les 431 modules, on en trouve 397 qui sont réutilisés, dont 358 dans au moins deux workflows. Les modules réutilisés sont présents en moyenne dans 6.9 workflows, et jusqu'à 46 workflows.
En Snakemake, la plateforme la plus importante de processeurs réutilisables est la plateforme *Snakemake wrappers repository* [9], la plateforme de partage "officielle" des *wrappers* Snakemake. On peut y trouver plus de 300 wrappers, correspondant à 146 outils et leurs options (de façon similaire aux modules nf-core, un wrapper par outil+option).
On retrouve 59 workflows faisant appel à des *wrappers* de la plateforme *Snakemake Wrapper Repository*.
Sur la totalité des *wrappers* disponibles, on en retrouve 44 dans nos workflows, dont 25 qui sont utilisés dans au moins deux workflows. Ces derniers sont présents en moyenne dans 4.3 workflows, et jusqu'à 21 workflows.

6. https://nf-co.re/modules
7. https://nf-co.re/modules/samtools_index
8. https://nf-co.re/modules/samtools_merge
9. https://snakemake-wrappers.readthedocs.io/en/stable/wrappers.html

4.6 . Identification des outils réutilisés

Nous nous intéressons maintenant à l'identification du type d'opérations effectuées par les utilisateurs dans leurs workflows. Pour identifier la réutilisation d'outils, nous comptons le nombre de fois où chaque outil est utilisé en considérant tous les processeurs des workflows.
Lors de l'étude des outils utilisés dans les processeurs, nous avons constaté que certains d'entre eux étaient largement utilisés, jusqu'à 2 841 fois, ce qui indique que de nombreux utilisateurs ont effectué des tâches très similaires.
Lorsque nous examinons la liste des outils les plus utilisés (top outils) pour Nextflow et Snakemake, nous constatons que 9 des 10 outils les plus utilisés sont communs à Nextflow et Snakemake. Dans la suite, nous nous concentrerons sur les 14 outils qui sont communs aux 20 outils les plus utilisés de Nextflow et Snakemake. Ces outils sont présentés dans le Tableau 4.2.

Tool	# NF	# SM	Catégorie
Samtools	2,841	2,045	GT
BEDTools [100]	384	603	GT
BCFtools [101]	929	360	GT
BWA	412	356	MAP
GATK	1067	269	GT
FastQC [102]	770	236	QC
Bowtie [103]	243	177	MAP
STAR [104]	234	152	MAP
MultiQC [105]	707	137	QC
Minimap2 [106]	149	137	MAP
Picard	269	137	GT
seqtk	96	94	GT
Cutadapt [107]	80	81	GT
QIIME [108]	207	90	DST

Table 4.2 – 14 outils communs entre les 20 principaux outils de Nextflow et les 20 principaux outils de Snakemake. # NF : Nombre de processeurs Nextflow dans lesquels ils apparaissent, # SM : Nombre de processeurs Snakemake dans lesquels ils apparaissent. Catégorie : (i) boîtes à outils génomiques (GT), (ii) alignement de séquences (MAP), (iii) outils de contrôle de qualité (QC) et (iv) outils spécifiques au domaine (DST).

Alors que certains outils sont utilisés pour le traitement générique de données bioinformatiques (par exemple, Samtools), certains sont des outils spécifiques à un domaine (par exemple, QIIME pour l'analyse microbiomique). Sans surprise, les boites à outils, les outils d'alignement et les outils de contrôle de qualité sont largement utilisés, car ils sont impliqués dans de nombreux workflows (même plusieurs

fois dans certains workflows) pour différents types d'analyse : *BWA* et *FastQC* sont utilisés avant la plupart des analyses de données de séquences, et *Samtools* est utilisé pour plusieurs tâches (par exemple, la conversion entre les formats d'alignement, le filtrage des reads) impliquées dans différents types d'analyses (par exemple, *RNA-Seq*, *SNP calling*).
En général, nous constatons que *i*) quelques outils sont largement réutilisés et *ii*) les outils largement utilisés dépendent peu du système de workflows utilisé, car les outils les plus utilisés sont presque les mêmes dans Nextflow et Snakemake.

4.7 . Structuration du jeu de données

Nous avons regroupé l'ensemble des informations extraites et calculées relatives aux workflows sous la forme d'une base de données relationnelle d'une part et sous la forme d'une base de connaissances d'autres part. La base de données relationnelle a été modélisée et remplie pour les workflows Nextflow lors d'un stage de M1, et ensuite étendue à Snakemake. Cette représentation est adaptée aux catégories de données que nous avons obtenu. Si Snakemake et Nextflow présentent des différences, notre analyse permet d'extraire des catégories d'annotations que nous pouvons comparer ou associer. .
La base de connaissances a ensuite été établie pour permettre de raisonner sémantiquement sur la fonction des workflows. Les workflows étant annotés avec des *topics* et *operations* EDAM, la traduction du jeu de données au format rdf permet notamment d'établir des métriques de similarité sémantiques sur ces données. Cette formalisation de la base facilite également son partage et l'interopérabilité avec d'autres sources dans le cadre du web sémantique.
Nous décrivons ces deux structures ci-après.

4.7.1 . Base de données relationnelle

Notre base de données relationnelle dont le schéma est décrit dans la figure 4.5 propose une représentation générique de workflows qui conviennent notamment à Snakemake et à Nextflow, sur la base des annotations que nous avons pu extraire pour chacun de ces systèmes grâce à notre outil.

Les spécifications de Nextflow et de Snakemake diffèrent suffisamment pour que les annotations extraites par l'outil possèdent quelques différences. Certaines annotations correspondent entre les deux langages (par exemple, "règle" Snakemake et "process" Nextflow représentent des processeurs). En revanche, d'autres annotations ne trouvent pas de correspondance directe : en Nextflow, les flux de données entre les processeurs sont spécifiés sous forme de "*channels*", tandis qu'en Snakemake, seuls les inputs et outputs sont spécifiés. De plus, ces *channels*

possèdent des noms et peuvent également comporter des "opérations" (par exemple, la duplication du flux de données pour relier une sortie de process à deux entrées de process.

Dans cette section, nous allons présenter les classes de la base de données.

workflow contient des annotations génériques sur le workflow, obtenues via *ParsWF*. Ce sont des métadonnées GitHub du dépôt du workflow, ainsi que les url du dépôt et du ou des fichiers. (2 932 entrées)
wf_nextflow et wf_snakemake regroupent les identifiants des workflows respectivement Nextflow et Snakemake. wf_nextflow contient la version du DSL (1 ou 2) du workflow. Ces tables contiennent également la structure au format *.dot* des workflows lorsqu'on a l'information. (1 675 et 1 257 entrées)
processor décrit les processeurs des workflows Nextflow et Snakemake. Ces annotations ont été obtenues via *ParsWF*. Les caractéristiques des processeurs Nextflow et Snakemake sont similaires mais les fonctionnalités proposées par ces deux systèmes sont différentes. On propose donc une modélisation générique qui tient compte de la diversité de fonctionnalités que nous avons pu détecter. Les processeurs contiennent tous une portion de "code", ici appelé script (un *shell*, un script python, des wrappers pour Snakemake,*etc.*). On renseigne le contenu de ce code dans *string_script* et on indique sa nature dans *language_script* et sa taille dans *nb_lines*. (49 321 entrées)
operation est spécifique aux workflows Nextflow. Ces annotations sont obtenues via *ParsWF*. Cette classe décrit les opérations appliquées aux données dans les arc Nextflow (filtrage, duplication du flux,...). Il n'existe pas d'équivalent en Snakemake car les flux de données sont spécifiés différemment. (4 339 entrées)
arc décrit les flux de données comme des arcs orientés, pour les workflows dont *ParsWF* a pu reconstituer la structure. Il contient l'id de la source dans *id_source* et celui de la destination dans *id_sink*. Ces éléments peuvent être des processeurs, des opérations, et des workflows, et peuvent appartenir à un workflow différent de l'arc. Il permet donc de prendre en compte les inclusions de processeurs et workflows externes (bien que nous n'avons pas à ce jour recherché ces informations). Pour Nextflow, les arcs sont des channels, et cette classe permet également d'inclure leurs annotations avec *type_channel* et *nom_channel*. (12 868 entrées)
git_person décrit les métadonnées GitHub associées aux utilisateurs ayant créé ou contribué à un workflow. Ces annotations sont issues de *CrawlWF* et décrit des attributs tels que (1 234 entrées)
bt_tool décrit les outils trouvés dans les workflows avec *WF2BT*. Nous avons spécifié une relation d'inclusion entre les outils et les processeurs, mais également

entre les workflows et les outils. (775 entrées)
edam_topic / edam_operation / edam_data contiennent les annotations EDAM des thèmes, opérations et type de données des inputs et outputs des outils provenant de bio.tools. Les liens entre les outils et ces tables sont établies via les relations "has_topic", "has_operation", et "has_data" (qui permet de spécifier si c'est un input ou un output de l'outil). (176/347/112 entrées)

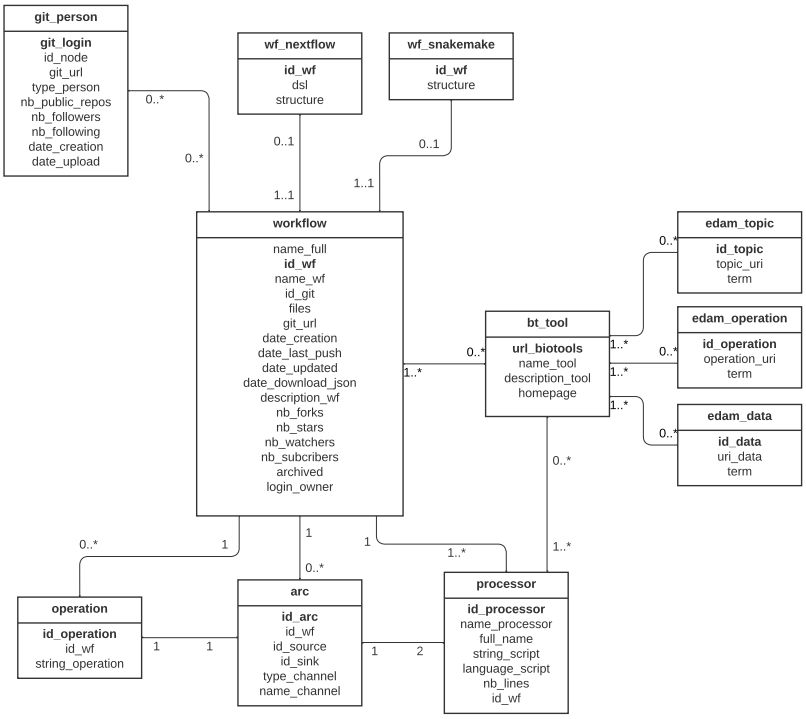

Figure 4.5 – Diagramme UML de la base de données des workflows Nextflow et Snakemake

Parmi les workflows analysés, nous avons extrait la structure de 207 workflows Snakemake et 753 workflows Nextflow (DSL1). Nous avons modélisé ces workflows sous forme de graphes annotés sémantiquement. Nous avons modélisé cette base sous une représentation simplifiée par rapport à la structure de la base de données relationnelle.

4.7.2 . Base de connaissances RDF

Les workflows et leurs processeurs ont aussi été décrits avec l'ontologie P-Plan. Cette ontologie permet de représenter des informations sur la provenance (provenance de données) et les plans d'exécution. Elle permet donc de décrire la

structure des workflows et de préciser l'ordre des processeurs notamment grâce à l'*object property* `pplan:isPrecededBy`.
Les processeurs ont été annotés selon les annotations EDAM (provenant de bio.tools) des outils qu'ils contiennent.
Les ontologies PPlan et Prov servent à décrire des annotations propres aux workflows : leur nom, leur auteur et leur url.

Étant donné que certaines des annotations que nous souhaitions incorporer ne sont pas couvertes par les ontologies existantes, nous avons dû créer des classes et des propriétés spécifiquement adaptées à nos besoins. Pour spécifier le système de gestion de workflows dans lequel le workflow est implémenté. Nous avons proposé la classe `r2p2o:workflow_system`, et la propriété `r2p2o:is_in_WFMS`. Pour représenter certaines annotations issues de GitHub, nous avons introduit les propriétés `r2p2o:has_nb_forks/r2p2o:has_nb_stars/r2p2o:has_nb_subscribers` pour les métriques "sociales" du dépôt GitHub du workflow, et `r2p2o:gitlogin` pour l'identifiant git de son auteur.

Ci-après 4.6, nous avons schématisé le graphe rdf d'un workflow exemple de deux processeurs, "workflow_1", tel qu'il serait représenté d'après notre modélisation.

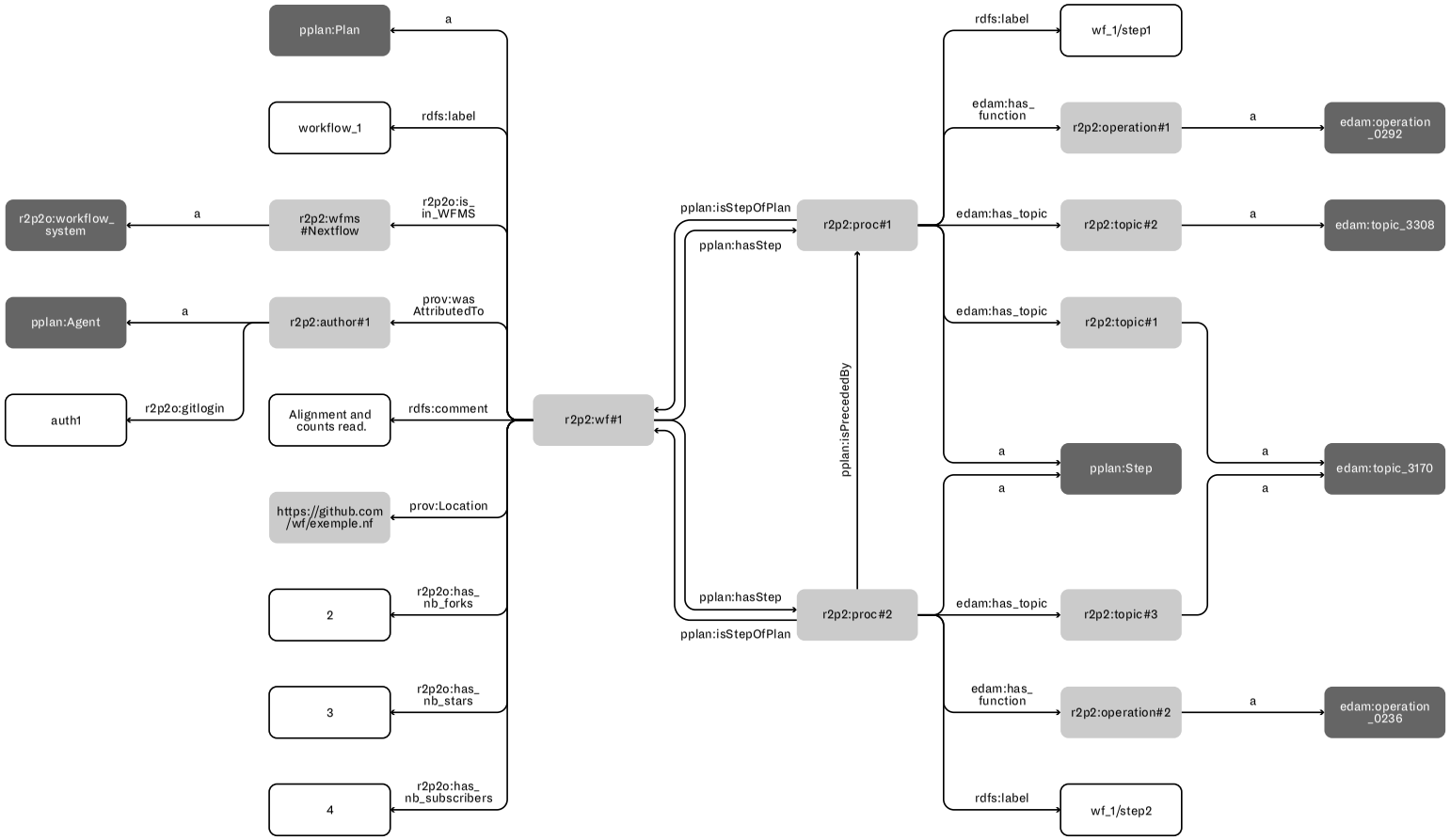

Figure 4.6 – Schéma d'un graphe RDF d'un exemple de workflow de deux processeurs modélisé selon notre représentation. Les classes sont représentées en gris foncé, les instances de classe en gris clair et les littéraux en blanc.

4.8 . Discussion

Bien que très similaires dans le principe, les spécifications de workflows Nextflow et Snakemake possèdent quelques différences. Une de ces différences majeures est la gestion des sous-workflows.
En Snakemake, il est possible d'inclure des processeurs et sous-workflows provenant de fichiers extérieurs au fichier de spécification depuis les débuts de Snakemake. Ces fichiers peuvent tous se trouver dans le même dossier (dans notre cas, sur le même dépôt Git) ou non. Certains de ces sous-workflows peuvent être exécutés de manière indépendante.
Ainsi, pour Snakemake, nous avons décidé de considérer **chaque fichier de spécification comme un workflow à part entière**.
En Nextflow, il n'a été possible d'inclure des processeurs et sous-workflows provenant de fichiers extérieurs qu'à partir de 2020, lors de la sortie de la version 2 du DSL de Nextflow (DSL2). Cependant, il n'est pas possible de déterminer si un workflow a été écrit en DSL1 ou en DSL2 en examinant uniquement la spécification.
Pour Nextflow, nous avons donc décidé de considérer que l'on a **un seul workflow par dépôt git**. Si il n'y a qu'un fichier de spécification dans le dépôt, on considère que c'est un workflow spécifié dans un seul fichier, et par défaut on déterminera qu'il est écrit en DSL1. Dans le cas où le dépôt contient plusieurs fichiers Nextflow, on considère que c'est un même workflow spécifié dans plusieurs fichiers, et qu'il est écrit en DSL2.

Cette différence influe sur le nombre de workflows trouvés et filtrés par notre outil dans chacun des deux systèmes :

- *CrawlWF* trouve :
    - pour Nextflow 1 790 workflows issus de 1 790 dépôts git ;
    - pour Snakemake 3 866 workflows issus de 1 908 dépôts git ;
- *ParsWF* conserve :
    - pour Nextflow 1 675 workflows issus de 1 675 dépôts git ;
    - pour Snakemake 2 946 workflows issus de 1 380 dépôts git ;
- *WF2BT* identifie :
    - pour Nextflow 1 267 workflows issus de 1 267 dépôts git ;
    - pour Snakemake 1 257 workflows issus de 708 dépôts git.

Malgré ces différences, les **workflows matched-tools** Nextflow sont sensiblement de la même taille que ceux en Snakemake : 15.8 processeurs pour Nextflow et 13.8 pour Snakemake. On peut donc considérer que ces workflows peuvent être

comparables.
On peut noter que parmi les **workflows matched-tools**, ceux écrits en Nextflow contiennent plus de **processeurs matched-tools** que pour Snakemake : 7.4 process contre 4.7 process par workflow en moyenne.

En outre, il n'est pas toujours possible d'établir une correspondance entre fichier (ou dossier) de spécification et workflow. Il peut y avoir plusieurs workflows distincts spécifiés dans un même fichier (ou dossier). Ou, comme énoncé précédemment, un seul workflow dans un même dossier.
Des informations de structure nous apporteraient des informations supplémentaires et affineraient la définition des workflows que nous avons établie. Elle permettrait notamment d'avoir une vue plus réaliste de quels ensembles de fichiers peuvent être considérés comme des workflows individuels, ou à l'inverse de quels fichiers contiennent plusieurs workflows. Cette définition nous permettrait notamment de nous rapprocher du véritable nombre de workflows Snakemake, que nous surestimons actuellement, de même pour le nombre de workflows Nextflow qui sont actuellement sous estimés.
On peut cependant noter que, dans le cas workflows Nextflow DSL1 (desquels nous pouvions obtenir la structure) nous n'avons détecté que très peu de cas où plusieurs workflows étaient définis dans le même fichier. Cette situation était en général de l'ordre de l'erreur (un processeur qui n'était relié à aucun autre, par exemple). Le cas contraire, celui de la présence de plusieurs workflows Nextflow considérés comme un unique workflow DSL2 a au contraire été observé à plusieurs reprises, comme par exemple dans le dépôt GitHub `https://github.com/ryanlayerlab/layer_lab_chco` qui contient 41 fichiers ".nf" correspondant à 7 workflows distincts.

Nextflow et Snakemake proposent tous deux de nombreuses fonctionnalités de "modularisation" (inclusion de sous-workflows et de processeurs), mais Snakemake se distingue particulièrement en offrant une gamme très large d'options dans ce domaine. Ces fonctionnalités visent à faciliter la réutilisation. Cependant, de notre point de vue, la dispersion des sources peut représenter un obstacle pour obtenir toutes les informations sur les composants d'un workflow, et par conséquent entraver la reconstruction de leur structure. Comme évoqué précédemment, les fichiers de spécification de sous-workflows ne sont pas toujours disponibles sur le même dépôt, et ils ne sont parfois pas du tout disponibles.

5 - Perspectives

Sommaire

Dans ce chapitre, nous présentons des perspectives ouvertes liées à nos contributions présentées dans le chapitre 4.
Dans un premier temps, nous synthétiserons les contributions de la thèse (5.1).
Dans un second temps, nous mettrons en évidence des étapes nécessaires pour favoriser l'adoption des workflows, notamment des systèmes Snakemake et Nextflow (5.2). Nous évoquerons des solutions pour améliorer la qualité des workflows (5.2.1), puis nous présenterons des perspectives nécessaires pour favoriser la réutilisation (5.2.2).

5.1 . Synthèse des contributions

Dans cette thèse, nous avons recensé les besoins des développeurs de workflows que nous avons associés à plusieurs étapes du cycle de vie du pipeline bioinformatique, et pour chaque étape, nous avons identifié des problèmes rencontrés et une vue d'ensemble d'une série de solutions. Plus précisément, nous avons d'abord présenté des éléments génériques de solutions, pouvant être utilisés dans le développement de tout type de pipeline. Nous nous sommes ensuite concentrés sur les systèmes de gestion de workflows scientifiques et en particulier sur la nouvelle génération de systèmes de workflows - Snakemake et Nextflow - qui offrent des avancées techniques significatives : le support natif des conteneurs aide à capturer l'environnement d'exécution, rendant les pipelines plus facilement reproductibles et donc plus faciles à partager et à réexécuter. Ces systèmes de gestion de workflows aident également à intégrer tous les composants d'un pipeline, tels que les outils, les scripts et les commandes bash, de manière transparente. De plus, la modularité des langages de workflows facilite l'isolement des étapes individuelles, rendant ainsi les pipelines plus faciles à comprendre pour les autres utilisateurs et favorisant également la réutilisation des étapes de workflows.
Nous avons proposé une méthode d'analyse de workflows à partir de leur spécification, et avons constitué un jeu de données de workflows annotés. Nous avons mené

une étude quantitative et qualitative de la réutilisation des workflows Snakemake et Nextflow à partir de ce jeu de données. Notre étude révèle qu'il existe déjà une réutilisation et fournit trois principales conclusions. Premièrement, entre un tiers et la moitié des propriétaires de workflows ont implémenté plusieurs workflows. Deuxièmement, nous avons trouvé des preuves de réutilisation dans le code source des processeurs. Troisièmement, nous avons mis en évidence un ensemble d'outils qui sont régulièrement utilisés parmi les pipelines implémentés sous forme de workflows. Des études de réutilisation antérieures ont été réalisées il y a dix ans sur les premiers systèmes. En particulier, [96] a étudié un ensemble de 898 workflows du système Taverna. Les conclusions de [96] diffèrent des nôtres. En ce qui concerne le partage, nous avons montré que les 10 propriétaires de workflows ayant écrit le plus de workflows en ont publié 15% (contre 62% auparavant), ce qui indique que l'utilisation des systèmes de workflows ne se limite plus à une communauté restreinte d'experts, mais à un éventail croissant de scientifiques. Par conséquent, les auteurs sont désormais plus disposés à partager et à réutiliser des workflows entre eux. Notre étude révèle enfin que l'existence de dépôts dédiés et organisés, tels que nf-core, contribue à promouvoir les pratiques de réutilisation.
Cependant, si une réutilisation existe, elle reste limitée. La réutilisation est un terme global qui correspond à plusieurs situations différentes, que nous avons listées dans nos besoins utilisateurs, comme :

- Besoin de réutiliser des fragments de pipelines pour compléter un pipeline (CU3)
- Besoin d'adapter un pipeline à de nouvelles données (CU5)
- Besoin de rechercher des pipelines pour comprendre les pratiques des autres utilisateurs (CU8)

Pour ces cas de figure, les solutions proposées sont incomplètes, notamment pour Snakemake et Nextflow qui sont de systèmes de plus en plus populaires. Pour faciliter la réutilisation, il faut s'assurer de la qualité des workflows, mais aussi permettre de les retrouver de façon efficace.

5.2 . Perspectives

Pour une adoption encore plus large des systèmes de workflows, les principales perspectives consistent à proposer des moyens de découvrir, de récupérer et de comparer des workflows. Ce point a été pris en compte dans la génération précédente de systèmes de workflows [9] et est encore souligné par des éditoriaux [5], des articles de la communauté très récents [109] et des articles de revue [6]. Les approches antérieures basées sur les systèmes de workflow Taverna (recherche de workflows [110], [99], indexation de workflows [111], recommandation de workflows

[87, 84, 112]) ne sont malheureusement pas bien adaptées à la nouvelle génération de systèmes et de dépôts où les workflows sont plus hétérogènes en termes de spécification et largement répartis sur le Web. Des défis techniques et algorithmiques restent à relever pour faire face à la nature distribuée, en constante évolution et en croissance continue des dépôts de workflows (basés sur git)[5, 109].

5.2.1 . Vers une meilleure qualité de workflows

Workflows FAIR

Un aspect clé à atteindre dans les années à venir pour une adoption plus large des workflows par les utilisateurs et pour accroître les pratiques de réutilisation des workflows est directement lié à la capacité d'identifier des workflows de haute qualité, c'est-à-dire *FAIR*. Les 'Principes directeurs FAIR pour la gestion et la conservation des données scientifiques' [113] fournissent des *guidelines* pour améliorer la recherche, l'accessibilité, l'interopérabilité et la réutilisabilité des ressources numériques. Appliquer les principes FAIR aux workflows scientifiques [10, 114] est particulièrement important car cela inclut la définition de métriques, par exemple, pour évaluer la capacité d'un workflow à être reproduit ou réutilisé, fournissant ainsi des informations clés sur la qualité aux (ré)utilisateurs de workflows.

Extraction de motifs de workflows

Un point clé pour faciliter la réutilisation et améliorer la qualité des workflows est la recherche de motifs fréquents, qui peut aider à identifier des bonnes pratiques.
Une solution utilisée pour trouver des motifs de workflows est la recherche de sous-graphes fréquents. Des travaux récents [115] ont proposé une méthode de recherche de motifs fréquents sur des workflows Galaxy. Leur étude porte sur 82 workflows dont ils parviennent à extraire 72 motifs de taille maximum 5.
Aucune méthode similaire n'existe pour Snakemake et Nextflow. Une limite notable d'application de telles méthodes à Snakemake et Nextflow est la définition de processeurs identiques. Dans Galaxy, tous les processeurs proviennent d'un *Toolshed* partagé, il est donc plus simple de savoir lesquels sont réutilisés dans plusieurs workflows.
Par ailleurs, la structure de ces workflows n'est pas une information facile d'accès, mais notre outil *ParsWF* permet de retrouver la structure de certains d'entre eux. La recherche de motifs dans des workflows Nextflow a fait l'objet d'un stage. Cette recherche a porté sur 556 workflows avec 15.93 noeuds et 19.18 arcs en moyenne par workflow, et nous avons fait usage de Gspan [116] pour retrouver des mo-

tifs. Nous avons considéré comme similaires les processeurs utilisant exactement les mêmes outils. Cependant, les résultats de cette étude ont été peu concluants, seuls 4 motifs intéressants (sur 24) ont été trouvés. Ces motifs étaient assez courts et étaient surtout des combinaisons de processeurs contenant les outils MultiQC et FastQC.
Une limite de notre démarche était la prise en compte des processeurs sur lesquels on n'avait pas d'informations : les processeurs sans outils (que l'on considérait comme tous uniques) et les opérations. En les enlevant des graphes, nous trouvions 14 motifs intéressants (sur 14). Pour une fouille de motifs plus efficace, il serait donc intéressant de rechercher de nouvelles façons d'identifier la réutilisation qui pourraient prendre en compte tous les types de processeurs. Une méthode se basant sur la similarité des codes, comme celle que nous avons proposée, pourrait donner de meilleurs résultats.
La recherche de motifs fréquents pourrait chercher à détecter non pas la combinaison de mêmes outils ou du même code, mais plus largement chercher à détecter quelles combinaisons d'opérations sont fréquentes. On pourrait utiliser des informations sémantiques pour établir des liens de similarité entre processeurs, qui peuvent être obtenues via *ParsWF*.

Similarité sémantique

Une autre catégorie de méthode de mesure de la similarité sont les méthodes sémantiques, qui représentent les workflows comme des graphes annotés sémantiquement et utilisent des techniques du web sémantique pour les comparer [117].
Un pré-requis majeur pour cette catégorie de mesure de la similarité est la qualité des annotations sémantiques.
Or, la plupart des annotations sémantiques que nous obtenons sur la fonction des workflows provient de bio.tools. Si la plupart des outils que nous avons détecté ont des topics et des opérations, seuls 23% d'entre eux ont un input ou un output. Par ailleurs, les topics et les annotations ne sont pas nécessairement précis. Les topics peuvent être des termes très génériques. Dans le cadre de "boîte à outils", les opérations ne sont pas liées à une ligne de commande, donc on associe toutes celles possibles à chaque appel de l'outil dans un processeur.
Il est aussi intéressant d'observer que seuls 40% des topics et 30% des opérations EDAM sont utilisés dans les annotations bio.tools.
Des travaux [118] ont proposé un cadre de représentation sémantique et des algorithmes de mesure de similarité sémantique entre workflows et processeurs. Cependant, leur méthodes ont été testées sur un ensemble de workflows réduit (20 workflows) qui ont été annotés manuellement. Ces méthodes sont prometteuses mais sont confrontées à plusieurs limites. Elles ne sont pas écrites pour une on-

tologie spécifique, mais considèrent que les données intermédiaires doivent être représentées dans le graphe du workflow (ce qui n'est pas pris en compte par toutes les ontologies, ni dans notre représentation de workflows, et qui n'est pas toujours une information donnée). Deuxièmement, ces méthodes ont été testées sur des ensembles très restreints, ce qui ne garantit pas qu'elles seront efficaces sur les ensembles de données bien plus importants aujourd'hui disponibles.
D'autres travaux plus récents [119] proposent d'établir des liens d'identité contextuels,*identyConTo*, entre des entités dans une base de connaissances rdf. En alternative au lien *sameas*, *identyConTo* permet de définir une similarité entre deux entités plus "flexible", en proposant un contexte dans lequel elles sont similaires (par exemple, un médicament et son générique sont similaires si on considère le principe actif et les dosages). Cette méthode peut être appliquée à des workflows, car elle permet de tenir compte de l'ordre des processeurs. Par ailleurs, elle permettrait d'apporter des informations sur les éléments déterminant la similarité entre deux entités. Exploiter de tels liens pourrait faciliter la recherche des workflows similaires selon une requête utilisateur plus rapidement qu'en faisant un calcul à chaque requête, même si calculer ces liens serait initialement très coûteux en temps et en ressources, surtout sur des ensembles de workflows importants. Cependant, l'adaptation de cette méthode aux workflows nécessiterait de définir des métriques de similarité plus adaptées aux processeurs et aux workflows.

5.2.2 . Complétion et Recherche de workflows

La conception et l'implémentation de workflows n'est pas une tâche aisée.
Si des solutions existent pour aider au choix d'outils spécifiques, un utilisateur qui veut faire des traitements plus complexes peut avoir besoin d'aide pour sélectionner un enchaînement d'outils, pour compléter un workflow existant ou pour créer un nouveau workflow.
De telles solutions existent dans des systèmes de workflows comme WINGS, mais sont spécifiques à ces systèmes.
APE in the wild [120] propose de générer, d'après des contraintes utilisateur (input, output, traitements désirés) un pipeline comprenant un ensemble d'outils issus de bio.tools sous forme de spécification CWL (et donc fonctionne pour n'importe quel système de workflows).
Cependant, le nombre de combinaisons possibles d'outils est très important, et les combinaisons peuvent être erronées (en contenant des enchaînements d'outils qui ne peuvent pas être combinés). APE in the wild ne permet pas de vérifier la pertinence des résultats proposés.
Pour ce faire, s'inspirer de pratiques utilisateurs en cours serait pertinent, notamment via la détection de motifs.
Les utilisateurs peuvent également chercher des workflows répondant à leurs be-

soins directement sur des plateformes de partage de workflows, comme présenté dans le chapitre 3 3.4.5. Dans un contexte de popularisation des workflows Snakemake et Nextflow, notamment, ces plateformes présentent de plus en plus de limites. La plateforme de workflows généraliste principale de workflows, successeure de myExperiment, peine à se faire adopter par la communauté : elle contient 402 workflows en Juillet 2023 contre une centaine en 2020, et un faible nombre de workflows Snakemake et Nextflow. Pourtant, de telles plateformes permettraient d'implémenter de nombreuses fonctionnalités utiles pour promouvoir la réutilisation et le partage (notamment en intégrant des informations sur les interactions sociales des utilisateurs [84] ou des fonctionnalités permettant de citer des workflows).
Les utilisateurs partagent pourtant leurs workflows sur GitHub, comme évoqué dans le chapitre 4 4.2. Snakemake workflow repository offre une perspective intéressante : c'est une plateforme de partage qui liste plus de 2000 dépôts GitHub contenant des workflows Snakemake. Seuls 167 d'entre eux sont des workflows "d'usage standard" proposés par des utilisateurs, et leur qualité (respect de bonnes pratiques établies par la communauté Snakemake) est évaluée et indiquée.
Cette plateforme aide les utilisateurs à trouver des workflows Snakemake plus facilement que sur GitHub, et a l'avantage de faciliter le processus de partage du pipeline (il "suffit" d'ajouter un fichier yaml et de respecter une certaine structure de fichiers). Cependant, les informations sur la fonction des workflows sont limitées et dépendantes de l'utilisateur (topics GitHub, README, description).
Notre outil d'annotation de workflows pourrait permettre à une plateforme sur le modèle de Snakemake Workflow repository d'améliorer les annotations des workflows indexés et notamment de proposer un moteur de recherche plus performant. Ce modèle de plateforme pourrait être plus en accord avec les pratiques utilisateurs.

Bibliographie

[1] Zekun Yin, Haidong Lan, Guangming Tan, Mian Lu, Athanasios V Vasilakos, and Weiguo Liu. Computing Platforms for Big Biological Data Analytics : Perspectives and Challenges. *Computational and Structural Biotechnology Journal*, 15 :403–411, 2017.

[2] Ben Langmead. Aligning short sequencing reads with bowtie. *Current protocols in bioinformatics*, 32(1) :11–7, 2010.

[3] Jeremy Leipzig. A review of bioinformatic pipeline frameworks. *Briefings in bioinformatics*, 18(3) :530–536, 2017.

[4] Monya Baker. Reproducibility crisis. *Nature*, 533(26) :353–66, 2016.

[5] Malcolm Atkinson, Sandra Gesing, Johan Montagnat, and Ian Taylor. Scientific workflows : Past , present and future. *Future Generation Computer Systems*, 75 :216–227, 2017.

[6] Laura Wratten, Andreas Wilm, and Jonathan Göke. Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. *Nature methods*, 18(10) :1161–1168, 2021.

[7] Marijn Van Vliet. Seven quick tips for analysis scripts in neuroimaging. *PLoS computational biology*, 16(3) :e1007358, 2020.

[8] Ola Spjuth, Erik Bongcam-Rudloff, Guillermo Carrasco Hernández, Lukas Forer, Mario Giovacchini, Roman Valls Guimera, Aleksi Kallio, Eija Korpelainen, Maciej M Kańduła, Milko Krachunov, et al. Experiences with workflows for automating data-intensive bioinformatics. *Biology direct*, 10(1) :1–12, 2015.

[9] Sarah Cohen-Boulakia and Ulf Leser. Search, adapt, and reuse : the future of scientific workflows. *ACM SIGMOD Record*, 40(2) :6–16, 2011.

[10] Carole Goble, Sarah Cohen-Boulakia, Stian Soiland-Reyes, Daniel Garijo, Yolanda Gil, Michael R. Crusoe, Kristian Peters, and Daniel Schober. Fair computational workflows. *Data Intelligence*, 2(1-2) :108–121, 2020.

[11] Sarah Cohen-Boulakia, Khalid Belhajjame, Olivier Collin, Jérôme Chopard, Christine Froidevaux, Alban Gaignard, Konrad Hinsen, Pierre Larmande, Yvan Le Bras, Frédéric Lemoine, Fabien Mareuil, Hervé Ménager, Christophe Pradal, and Christophe Blanchet. Scientific workflows for computational reproducibility in the life sciences : Status, challenges and opportunities. *Fut Gen Comput Systems*, 75 :284–298, 2017.

[12] Christophe Pradal, Simon Artzet, Jérôme Chopard, Dimitri Dupuis, Christian Fournier, Michael Mielewczik, Vincent Nègre, Pascal Neveu, Didier Parigot, Patrick Valduriez, and Sarah Cohen Boulakia. Infraphenogrid : A scientific workflow infrastructure for plant phenomics on the grid. *Future Generation Comp. Syst.*, 67 :341–353, 2017.

[13] Enis Afgan, Anton Nekrutenko, Bjórn A Grüning, Daniel Blankenberg, Jeremy Goecks, Michael C Schatz, Alexander E Ostrovsky, Alexandru Mahmoud, Andrew J Lonie, Anna Syme, Anne Fouilloux, Anthony Bretaudeau, Anton Nekrutenko, Anup Kumar, Arthur C Eschenlauer, Assunta D DeSanto, Aysam Guerler, Beatriz Serrano-Solano, Bérénice Batut, Björn A Grüning, Bradley W Langhorst, Bridget Carr, Bryan A Raubenolt, Cameron J Hyde, Catherine J Bromhead, Christopher B Barnett, Coline Royaux, Cristóbal Gallardo, Daniel Blankenberg, Daniel J Fornika, Dannon Baker, Dave Bouvier, Dave Clements, David A de Lima Morais, David Lopez Tabernero, Delphine Lariviere, Engy Nasr, Enis Afgan, Federico Zambelli, Florian Heyl, Fotis Psomopoulos, Frederik Coppens, Gareth R Price, Gianmauro Cuccuru, Gildas Le Corguillé, Greg Von Kuster, Gulsum Gudukbay Akbulut, Helena Rasche, Hans-Rudolf Hotz, Ignacio Eguinoa, Igor Makunin, Isuru J Ranawaka, James P Taylor, Jayadev Joshi, Jennifer Hillman-Jackson, Jeremy Goecks, John M Chilton, Kaivan Kamali, Keith Suderman, Krzysztof Poterlowicz, Le Bras Yvan, Lucille Lopez-Delisle, Luke Sargent, Madeline E Bassetti, Marco Antonio Tangaro, Marius van den Beek, Martin Čech, Matthias Bernt, Matthias Fahrner, Mehmet Tekman, Melanie C Föll, Michael C Schatz, Michael R Crusoe, Miguel Roncoroni, Natalie Kucher, Nate Coraor, Nicholas Stoler, Nick Rhodes, Nicola Soranzo, Niko Pinter, Nuwan A Goonasekera, Pablo A Moreno, Pavankumar Videm, Petera Melanie, Pietro Mandreoli, Pratik D Jagtap, Qiang Gu, Ralf J M Weber, Ross Lazarus, Ruben H P Vorderman, Saskia Hiltemann, Sergey Golitsynskiy, Shilpa Garg, Simon A Bray, Simon L Gladman, Simone Leo, Subina P Mehta, Timothy J Griffin, Vahid Jalili, Vandenbrouck Yves, Victor Wen, Vijay K Nagampalli, Wendi A Bacon, Willem de Koning, Wolfgang Maier, and Peter J Briggs. The galaxy platform for accessible, reproducible and collaborative biomedical analyses : 2022 update. *Nucleic Acids Research*, 50(W1) :W345–W351, April 2022.

[14] Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19) :2520–2522, 2012.

[15] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature biotechnology*, 35(4) :316, 2017.

[16] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan R Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex R Hardisty, Abraham Nieva de la Hidalga, Maria P Balcazar Vargas, Shoaib Sufi, and Carole A Goble. The Taverna workflow suite : designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Res.*, 41(Webserver-Issue) :557–561, 2013.

[17] Marine Djaffardjy, George Marchment, Clémence Sebe, Raphael Blanchet, Khalid Bellajhame, Alban Gaignard, Frédéric Lemoine, and Sarah Cohen-Boulakia. Developing and reusing bioinformatics data analysis pipelines using scientific workflow systems. *Computational and Structural Biotechnology Journal*, 21, 03 2023.

[18] Romain Bourcier, Solena Le Scouarnec, Stéphanie Bonnaud, Matilde Karakachoff, Emmanuelle Bourcereau, Sandrine Heurtebise-Chrétien, Celine Menguy, Christian Dina, Floriane Simonet, Alexis Moles, Cédric Lenoble, Pierre Lindenbaum, Stéphanie Chatel, Bertrand Isidor, Emmanuelle Génin, Jean-François Deleuze, Jean-Jacques Schott, Hervé Le Marec, Gervaise Loirand, Hubert Desal, and Richard Redon. Rare coding variants in angptl6 are associated with familial forms of intracranial aneurysm. *American journal of human genetics*, 102 1 :133–141, 2018.

[19] Olivia Rousseau, Matilde Karakachoff, Alban Gaignard, Lise Bellanger, Philippe Bijlenga, Pacôme Constant Dit Beaufils, Vincent L'Allinec, Olivier Levrier, Pierre Aguettaz, Jean-Philippe Desilles, Caterina Michelozzi, Gaultier Marnat, Anne-Clémence Vion, Gervaise Loirand, Hubert Desal, Richard Redon, Pierre-Antoine Gourraud, and Romain Bourcier. Location of intracranial aneurysms is the main factor associated with rupture in the ican population. *Journal of Neurology, Neurosurgery & Psychiatry*, 92(2) :122–128, 2021.

[20] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16) :2078–2079, 2009.

[21] Eric Charpentier, Marine Cornec, Solenne Dumont, Dimitri Meistermann, Philippe Bordron, Laurent David, Richard Redon, Stéphanie Bonnaud, and Audrey Bihouée. 3'rna sequencing for robust and low-cost gene expression profiling. *protocol exchange*, 2021.

[22] Marie Morel, Frédéric Lemoine, Anna Zhukova, and Olivier Gascuel. Accurate detection of convergent mutations in large protein alignments with condor. *bioRxiv*, 2022.

[23] Jörg Fehr, Jan Heiland, Christian Himpe, and Jens Saak. Best practices for replicability, reproducibility and reusability of computer-based experiments exemplified by model reduction software. *AIMS Mathematics*, 1 :261–, 09 2016.

[24] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *bioinformatics*, 25(14) :1754–1760, 2009.

[25] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, et al. The genome analysis toolkit : a mapreduce framework for analyzing next-generation dna sequencing data. *Genome research*, 20(9) :1297–1303, 2010.

[26] Broad Institute. Picard tools. http://broadinstitute.github.io/picard/, (Accessed : 2022/10/24 ; version 2.27.4).

[27] Adam Rule, Amanda Birmingham, Cristal Zuniga, Ilkay Altintas, Shih-Cheng Huang, Rob Knight, Niema Moshiri, Mai H Nguyen, Sara Brin Rosenthal, Fernando Pérez, et al. Ten simple rules for writing and sharing computational analyses in jupyter notebooks, 2019.

[28] Aziz Nanthaamornphong and Jeffrey C Carver. Test-driven development in scientific software : a survey. *Software Quality Journal*, 25 :343–372, 2017.

[29] Jamie J Alnasir. Fifteen quick tips for success with hpc, ie, responsibly bashing that linux cluster. *PLOS Computational Biology*, 17(8) :e1009207, 2021.

[30] D Stott Parker, Michael M Gorlick, and Christopher J Lee. Evolving from bioinformatics in-the-small to bioinformatics in-the-large. *OMICS A Journal of Integrative Biology*, 7(1) :37–48, 2003.

[31] Don Brown, Bakr A Hassan, and Eric Hjelmfelt. A basic review of mathematica. *International journal of mathematical education in science and technology*, 22(2) :207–221, 1991.

[32] Friedrich Leisch. Sweave : Dynamic generation of statistical reports using literate data analysis. In *COMPSTAT : Proceedings in computational statistics*, pages 575–580. Springer, 2002.

[33] Yihui Xie. knitr : a comprehensive tool for reproducible research in r. In *Implementing reproducible research*, pages 3–31. Chapman and Hall/CRC, 2018.

[34] Benjamin Baumer and Dana Udwin. R markdown. *Wiley Interdisciplinary Reviews : Computational Statistics*, 7(3) :167–177, 2015.

[35] Benjamin Ragan-Kelley, Carol Willing, F Akici, D Lippa, D Niederhut, and M Pacer. Binder 2.0-reproducible, interactive, sharable environments for science at scale. In *Proceedings of the 17th python in science conference*, pages 113–120. F. Akici, D. Lippa, D. Niederhut, and M. Pacer, eds., 2018.

[36] Bernadette M Randles, Irene V Pasquetto, Milena S Golshan, and Christine L Borgman. Using the jupyter notebook as a tool for open science : An empirical study. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 1–2. IEEE, 2017.

[37] Jiawei Wang, Tzu-yang Kuo, Li Li, and Andreas Zeller. Assessing and restoring reproducibility of jupyter notebooks. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 138–149, 2020.

[38] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. Understanding and improving the quality and reproducibility of jupyter notebooks. *Empirical Software Engineering*, 26(4) :65, 2021.

[39] Tom Oinn, Mark Greenwood, Matthew J Addis, M Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, DJ Marvin, et al. Taverna : lessons in creating a workflow environment for the life sciences. *Journal of Concurrency and Computation : Practice and experience*, 2004.

[40] Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos Eduardo Scheidegger, Cláudio T Silva, and Huy T Vo. Managing the evolution of dataflows with vistrails. In *22nd International Conference on Data Engineering Workshops (ICDEW'06)*, pages 71–71. IEEE, 2006.

[41] Ustun Yildiz, Adnene Guabtni, and Anne H.H. Ngu. Business versus scientific workflows : A comparative study. In *2009 Congress on Services - I*, pages 340–343, 2009.

[42] Marine Djaffardjy, George Marchment, Clémence Sebe, Raphael Blanchet, Khalid Bellajhame, Alban Gaignard, Frédéric Lemoine, and Sarah Cohen-Boulakia. Developing and reusing bioinformatics data analysis pipelines using scientific workflow systems. *Computational and Structural Biotechnology Journal*, 2023.

[43] Levin Clement, Dynomant Emeric, Mouchard Laurent, Landsman David, Hovig Eivind, Vlahovicek Kristian, et al. A data-supported history of bioinformatics tools. *arXiv preprint arXiv :1807.06808*, 2018.

[44] Jiten Bhagat, Franck Tanoh, Eric Nzuobontane, Thomas Laurent, Jerzy Orlowski, Marco Roos, Katy Wolstencroft, Sergejs Aleksejevs, Robert Stevens,

Steve Pettifer, et al. Biocatalogue : a universal catalogue of web services for the life sciences. *Nucleic acids research*, page gkq394, 2010.

[45] Vincent J Henry, Anita E Bandrowski, Anne-Sophie Pepin, Bruno J Gonzalez, and Arnaud Desfeux. Omictools : an informative directory for multi-omic data analysis. *Database*, 2014, 2014.

[46] Jon Ison, Kristoffer Rapacki, Hervé Ménager, Matúš Kalaš, Emil Rydza, Piotr Chmura, Christian Anthon, Niall Beard, Karel Berka, Dan Bolser, et al. Tools and data services registry : a community effort to document bioinformatics resources. *Nucleic acids research*, 44(D1) :D38–D47, 2016.

[47] Jon Ison, Hans Ienasescu, Emil Rydza, Piotr Chmura, Kristoffer Rapacki, Alban Gaignard, Veit Schwämmle, Jacques Van Helden, Matúš Kalaš, and Hervé Ménager. biotoolsschema : a formalized schema for bioinformatics software description. *GigaScience*, 10(1) :giaa157, 2021.

[48] Jon et al. Ison. Edam : an ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics*, 29(10) :1325–1332, 2013.

[49] Damian Smedley, Syed Haider, Benoit Ballester, Richard Holland, Darin London, Gudmundur Thorisson, and Arek Kasprzyk. Biomart–biological queries made easy. *BMC genomics*, 10(1) :1–12, 2009.

[50] Paolo Romano, Rosalba Giugno, and Alfredo Pulvirenti. Tools and collaborative environments for bioinformatics research. *Briefings in bioinformatics*, 12(6) :549–561, 2011.

[51] Jia Zhang. Co-taverna : A tool supporting collaborative scientific workflows. In *2010 IEEE International Conference on Services Computing*, pages 41–48. IEEE, 2010.

[52] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. A large-scale study about quality and reproducibility of jupyter notebooks. In *2019 IEEE/ACM 16th international conference on mining software repositories (MSR)*, pages 507–517. IEEE, 2019.

[53] Susan B Davidson, Sarah Cohen Boulakia, Anat Eyal, Bertram Ludäscher, Timothy M McPhillips, Shawn Bowers, Manish Kumar Anand, and Juliana Freire. Provenance in scientific workflow systems. *IEEE Data Eng. Bull.*, 30(4) :44–50, 2007.

[54] Farah Zaib Khan, Stian Soiland-Reyes, Richard O Sinnott, Andrew Lonie, Carole Goble, and Michael R Crusoe. Sharing interoperable workflow provenance : A review of best practices and their practical application in cwlprov. *GigaScience*, 8(11) :giz095, 2019.

[55] Luc Moreau, Juliana Freire, Joe Futrelle, Robert E McGrath, Jim Myers, and Patrick Paulson. The open provenance model : An overview. In *International provenance and annotation workshop*, pages 323–326. Springer, 2008.

[56] Luc Moreau, Paul Groth, James Cheney, Timothy Lebo, and Simon Miles. Web Semantics : Science , Services and Agents on the World Wide Web The rationale of PROV. *Web Semantics : Science, Services and Agents on the World Wide Web*, 35 :235–257, 2015.

[57] Flavio Costa, Vítor Silva, Daniel de Oliveira, Kary A. C. S. Ocaña, Eduardo S. Ogasawara, Jonas Dias, and Marta Mattoso. Capturing and querying workflow runtime provenance with PROV : a practical approach. In *Joint 2013 EDBT/ICDT Conferences, EDBT/ICDT '13, Genoa, Italy, March 22, 2013, Workshop Proceedings*, pages 282–289, 2013.

[58] Peter Sefton, Eoghan Ó Carragáin, Carole Goble, and Stian Soiland-Reyes. Introducing RO-Crate : research object data packaging. In *eResearch Australasia 2019*, 2019.

[59] Marco S Nobile, Paolo Cazzaniga, Andrea Tangherloni, and Daniela Besozzi. Graphics processing units in bioinformatics, computational biology and systems biology. *Briefings in bioinformatics*, 18(5) :870–885, 2017.

[60] Albert Reuther, Chansup Byun, William Arcand, David Bestor, Bill Bergeron, Matthew Hubbell, Michael Jones, Peter Michaleas, Andrew Prout, Antonio Rosa, and Jeremy Kepner. Highlights of Scalable System Scheduling for HPC and Big Data article Albert Reuther. *J. Parallel Distrib. Comput.*, 2017.

[61] Morris A. Jette, Andy B. Yoo, and Mark Grondona. Slurm : Simple linux utility for resource management. In *In Lecture Notes in Computer Science : Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, pages 44–60. Springer-Verlag, 2002.

[62] Hanhua Feng, Vishal Misra, and Dan Rubenstein. Pbs : a unified priority-based scheduler. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 203–214, 2007.

[63] Carl Boettiger. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1) :71–79, 2015.

[64] Michael Eder. Hypervisor-vs. container-based virtualization. *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, 1, 2016.

[65] Stephen R Piccolo and Michael B Frampton. Tools and techniques for computational reproducibility. *GigaScience*, 5(1) :s13742–016, 2016.

[66] Maximilian Hanussek, Felix Bartusch, and Jens Krüger. *Performance and scaling behavior of bioinformatic applications in virtualization environments to create awareness for the efficient use of compute resources*, volume 17. Public Library of Science San Francisco, CA USA, 2021.

[67] Marek Moravcik, Pavel Segec, Martin Kontsek, Jana Uramova, and Jozef Papan. Comparison of lxc and docker technologies. In *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 481–486. IEEE, 2020.

[68] Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1) :71–79, 2015.

[69] Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. Singularity : Scientific containers for mobility of compute. *PloS one*, 12(5) :e0177459, 2017.

[70] Viktória Spišaková, Lukáš Hejtmánek, and Jakub Hynšt. Nextflow in Bioinformatics : Executors Performance Comparison Using Genomics Data. *Future Generation Computer Systems*, 142 :328–339, 2023.

[71] Rui Shu, Xiaohui Gu, and William Enck. A study of security vulnerabilities on docker hub. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 269–280, 2017.

[72] François Moreews, Olivier Sallou, Hervé Ménager, Cyril Monjeaud, Christophe Blanchet, Olivier Collin, et al. Bioshadock : a community driven bioinformatics shared docker-based tools registry. *F1000Research*, 4, 2015.

[73] Felipe da Veiga Leprevost, Björn A Grüning, Saulo Alves Aflitos, Hannes L Röst, Julian Uszkoreit, Harald Barsnes, Marc Vaudel, Pablo Moreno, Laurent Gatto, Jonas Weber, et al. Biocontainers : an open-source and community-driven framework for software standardization. *Bioinformatics*, 33(16) :2580–2582, 2017.

[74] Björn Grüning, Ryan Dale, Andreas Sjödin, Brad A Chapman, Jillian Rowe, Christopher H Tomkins-Tinch, Renan Valieris, and Johannes Köster. Bioconda : sustainable and comprehensive software distribution for the life sciences. *Nature methods*, 15(7) :475–476, 2018.

[75] Ludovic Courtès. Functional package management with guix. *arXiv preprint arXiv :1305.4584*, 2013.

[76] Philip A. Ewels, Alexander Peltzer, Sven Fillinger, Harshil Patel, Johannes Alneberg, Andreas Wilm, Maxime Ulysse Garcia, Paolo Di Tommaso, and Sven Nahnsen. The nf-core framework for community-curated bioinformatics pipelines. *Nature Biotechnology 2020 38 :3*, 38(3) :276–278, feb 2020.

[77] usegalaxy. Extracting workflows from histories. https://training.galaxyproject.org/training-material/topics/galaxy-interface/tutorials/history-to-workflow/tutorial.html [Accessed : 20/07/23].

[78] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. pytest, 2004.

[79] Gideon Juve, Ann L. Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Comp. Syst.*, 29(3) :682–692, 2013.

[80] Peter Amstutz, Michael R Crusoe, Nebojša Tijanić, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Hervé Ménager, Maya Nedeljkovich, et al. Common workflow language, v1. 0, 2016.

[81] Laura Wratten, Andreas Wilm, and Jonathan Göke. Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. *Nature Methods*, 18(10) :1161–1168, 2021.

[82] Thomas Cokelaer, Dimitri Desvillechabrol, Rachel Legendre, and Mélissa Cardon. 'sequana' : a set of snakemake ngs pipelines. *Journal of Open Source Software*, 2(16) :352, 2017.

[83] Carole Goble, Stian Soiland-Reyes, Finn Bacall, Stuart Owen, Alan Williams, Ignacio Eguinoa, Bert Droesbeke, Simone Leo, Luca Pireddu, Laura Rodríguez-Navas, et al. Implementing fair digital objects in the eosc-life workflow collaboratory. *Zenodo*, 2021.

[84] Carole A Goble, Jiten Bhagat, Sergejs Aleksejevs, Don Cruickshank, Danius Michaelides, David Newman, Mark Borkum, Sean Bechhofer, Marco Roos, Peter Li, et al. myexperiment : a repository and social network for the sharing of bioinformatics workflows. *Nucleic acids research*, 38(suppl 2) :W677–W682, 2010.

[85] Phillip Mates, Emanuele Santos, Juliana Freire, and Cláudio T Silva. Crowdlabs : Social analysis and visualization for the sciences. In *Scientific and Statistical Database Management*, pages 555–564. Springer, 2011.

[86] Vladimir Korkhov, Dagmar Krefting, Johan Montagnat, T Truong Huu, Tamas Kukla, Gabor Terstyanszky, David Manset, Matthan Caan, and Silvia Olabarriaga. Shiwa workflow interoperability solutions for neuroimaging data analysis. *Stud Health Technol Inform*, 175, 2012.

[87] David De Roure and Carole Goble. Lessons from myExperiment : Two insights into emerging e-Research practice. *UK eScience All Hands Meeting 2009*, pages 6–8, 2009.

[88] J. Trant and archives informatics. Studying social tagging and folksonomy : A review and framework. *Journal of Digital Information ; Vol 10, No 1 (2009)*, 10, 01 2009.

[89] Johannes Starlinger, Bryan Brancotte, Sarah Cohen-Boulakia, and Ulf Leser. Similarity search for scientific workflows. *Proc. VLDB Endow.*, 7(12) :1143–1154, aug 2014.

[90] Antoon Goderis, Paul Fisher, Andrew Gibson, Franck Tanoh, Katy Wolstencroft, David De Roure, and Carole Goble. Benchmarking workflow discovery : a case study from bioinformatics. *Concurrency and Computation : Practice and Experience*, 21(16) :2052–2069, 2009.

[91] Antoon Goderis, Peter Li, and Carole Goble. Workflow discovery : The problem, a case study from e-science and a graph-based solution. In *Proceedings - ICWS 2006 : 2006 IEEE International Conference on Web Services*, pages 312–319, 01 2006.

[92] Bruno T Messmer and Horst Bunke. Efficient subgraph isomorphism detection : a decomposition approach. *IEEE transactions on knowledge and data engineering*, 12(2) :307–323, 2000.

[93] X DONG, A HALEVY, J MADHAVAN, E NEMES, and J ZHANG. Similarity Search for Web Services. *Proceedings 2004 VLDB Conference*, pages 372–383, 2004.

[94] Daniel Silva Junior, Esther Pacitti, Aline Paes, and Daniel de Oliveira. Provenance-and machine learning-based recommendation of parameter values in scientific workflows. *PeerJ Computer Science*, 7 :1–46, 2021.

[95] Trung Dong Huynh and Luc Moreau. Provstore : A public provenance repository. In Bertram Ludäscher and Beth Plale, editors, *Provenance and Annotation of Data and Processes*, pages 275–277, Cham, 2015. Springer International Publishing.

[96] Johannes Starlinger, Sarah Cohen Boulakia, and Ulf Leser. (re)use in public scientific workflow repositories. In *Scientific and Statistical Database Management - 24th International Conference, SSDBM 2012, Chania, Crete, Greece, June 25-27, 2012. Proceedings*, pages 361–378, 2012.

[97] Jon Ison, Hans Ienasescu, Piotr Chmura, Emil Rydza, Herve Menager, Matuš Kalaš, Veit Schwammle, Bjorn Gruning, Niall Beard, Rodrigo Lopez, Severine Duvaud, Heinz Stockinger, Bengt Persson, Radka Svobodova Vařekova, Tomaš Raček, Jiři Vondrašek, Hedi Peterson, Ahto Salumets, Inge Jonassen, Rob Hooft, Tommi Nyronen, Alfonso Valencia, Salvador Capella, Josep Gelpi, Federico Zambelli, Babis Savakis, Brane Leskošek, Kristoffer Rapa-

cki, Christophe Blanchet, Rafael Jimenez, Arlindo Oliveira, Gert Vriend, Olivier Collin, Jacques Van Helden, Peter Løngreen, and Søren Brunak. The bio.tools registry of software tools and data resources for the life sciences. *Genome Biology*, 20(1) :1–4, 2019.

[98] Matija Novak, Mike Joy, and Dragutin Kermek. Source-code similarity detection and detection tools used in academia : A systematic review. *ACM Trans. Comput. Educ.*, 19(3), may 2019.

[99] Johannes Starlinger, Bryan Brancotte, Sarah Cohen-Boulakia, and Ulf Leser. Similarity search for scientific workflows. *Proceedings of the VLDB Endowment*, 7(12) :1143–1154, 2014.

[100] Aaron R Quinlan and Ira M Hall. Bedtools : a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6) :841–842, 2010.

[101] Petr Danecek, James K Bonfield, Jennifer Liddle, John Marshall, Valeriu Ohan, Martin O Pollard, Andrew Whitwham, Thomas Keane, Shane A McCarthy, Robert M Davies, and Heng Li. Twelve years of SAMtools and BCFtools. *GigaScience*, 10(2), 02 2021. giab008.

[102] Simon Andrews, Felix Krueger, Anne Segonds-Pichon, Laura Biggins, Christel Krueger, and Steven Wingett. FastQC. Babraham Institute, January 2012.

[103] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4) :357–359, 2012.

[104] Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. Star : ultrafast universal rna-seq aligner. *Bioinformatics*, 29(1) :15–21, 2013.

[105] Philip Ewels, Måns Magnusson, Sverker Lundin, and Max Käller. Multiqc : summarize analysis results for multiple tools and samples in a single report. *Bioinformatics*, 32(19) :3047–3048, 2016.

[106] Heng Li. Minimap2 : pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18) :3094–3100, 2018.

[107] Marcel Martin. Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet. journal*, 17(1) :10–12, 2011.

[108] J Gregory Caporaso, Justin Kuczynski, Jesse Stombaugh, Kyle Bittinger, Frederic D Bushman, Elizabeth K Costello, Noah Fierer, Antonio Gonzalez Peña, Julia K Goodrich, Jeffrey I Gordon, et al. Qiime allows analysis of high-throughput community sequencing data. *Nature methods*, 7(5) :335–336, 2010.

[109] Rafael Ferreira Da Silva, Henri Casanova, Kyle Chard, Ilkay Altintas, Rosa M Badia, Bartosz Balis, Taina Coleman, Frederik Coppens, Frank Di Natale, Bjoern Enders, et al. A community roadmap for scientific workflows research and development. In *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*, pages 81–90. IEEE, 2021.

[110] Antoon Goderis, David De Roure, Carole Goble, Jiten Bhagat, Don Cruickshank, Paul Fisher, Danius Michaelides, and Franck Tanoh. Discovering scientific workflows : The myexperiment benchmarks. 2008.

[111] Julia Stoyanovich, Ben Taskar, and Susan Davidson. Exploring repositories of scientific workflows. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2010.

[112] Ahmed Halioui, Tomas Martin, Petko Valtchev, and Abdoulaye Baniré Diallo. Ontology-based workflow pattern mining : Application to bioinformatics expertise acquisition. *Proceedings of the ACM Symposium on Applied Computing*, Part F128005 :824–827, 2017.

[113] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3, 2016.

[114] Remzi Celebi, Joao Rebelo Moreira, Ahmed A. Hassan, Sandeep Ayyar, Lars Ridder, Tobias Kuhn, and Michel Dumontier. Towards FAIR protocols and workflows : The OpenPREDICT use case. *PeerJ Computer Science*, 6 :1–29, 2020.

[115] CR Wijesinghe and AR Weerasinghe. Mining frequent patterns in bioinformatics workflows. *Int. J. Bioscience, Biochemistry, Bioinformatics*, 10(4) :161–169, 2020.

[116] Xifeng Yan and Jiawei Han. gspan : Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724. IEEE, 2002.

[117] Zhangbing Zhou, Zehui Cheng, Liang-Jie Zhang, Walid Gaaloul, and Ke Ning. Scientific workflow clustering and recommendation leveraging layer hierarchical analysis. *IEEE Transactions on Services Computing*, 11(1) :169–183, 2016.

[118] Ralph Bergmann and Yolanda Gil. Similarity assessment and efficient retrieval of semantic workflows. *Information Systems*, 40 :115–127, 2014.

[119] Joe Raad, Nathalie Pernelle, and Fatiha Saïs. Detection of contextual identity links in a knowledge base. In *Proceedings of the knowledge capture conference*, pages 1–8, 2017.

[120] Vedran Kasalica, Veit Schwämmle, Magnus Palmblad, Jon C. Ison, and Anna-Lena Lamprecht. Ape in the wild : Automated exploration of proteomics workflows in the bio.tools registry. *Journal of Proteome Research*, 20 :2157 – 2165, 2021.