Université de Rennes

istic Informatique Electronique

Rapport fin de parcours projet BIO4T

LEBOEUF Alexis, GUINCHE Mathéo,
CHEVALIER Kilian, MBASSI ATANGANA Blaise,
MAWUVI Kodjo David Alexis

M1 IL - 2024-2025

Table des Matières

1 Introduction

1.1 Contexte

La BioInformatique est une interdiscipline à la frontière de la biologie, de l'informatique et des mathématiques qui permet d'analyser toute une banque d'informations contenue dans les cellules vivantes sous forme de séquences nucléiques ou protéiques. Elle répond à un besoin croissant de traiter de vastes quantités de données générées par des technologies modernes telles que le séquençage génomique, la protéomique, la transcriptomique, ainsi que par l'étude des interactions moléculaires et des réseaux biologiques.

Les techniques utilisées incluent des approches statistiques, algorithmiques et computationnelles pour traiter tout ce flux de données. Parmi ces méthodes clés, on retrouve l'alignement de séquences, un procédé consistant à comparer des séquences génétiques ou protéiques dans le but d'identifier des similarités et des relations évolutives.

Une autre technique importante est le scaffolding encore appelé assemblage de génomes qui est une technique d'assemblage de fragments courts de séquences génétiques dans l'intention de reconstruire un génome plus précis et complet, un peu à la manière d'un puzzle où les pièces manquantes sont connectées à partir d'informations supplémentaires.

La bio-informatique est une discipline en constante évolution, en raison des défis et des nouveaux problèmes posés par la biologie qui exigent des approches innovantes et une adaptation continue des méthodes scientifiques.

Dans le domaine de la recherche, le travail de collaboration et d'innovation est central, et les avancées scientifiques doivent être accessibles facilement à la communauté scientifique, dans une philosophie de science ouverte, et d'une meilleure avancée mondiale.

Dans ce contexte, l'Union européenne a créé ELIXIR, qui est une organisation intergouvernementale. Son travail est de rassembler les ressources en lien avec les sciences de la vie. Dans ce but, elle a créé biotools, une plateforme répertoriant plus de 30 000 outils. En pratique, les outils sont rarement utilisés seuls, mais sous la forme d'une suite d'outils utilisés et de manipulations réalisées.

Les outils peuvent être intégrés dans un workflow. Dans le contexte de la bioinformatique et des sciences de la vie, un workflow est une séquence d'étapes structurées et organisées qui permet de traiter ou d'analyser des données (figure 1). L'automatisation des workflows permet un gain de temps considérable pour des personnes non spécialistes de la bio-informatique.

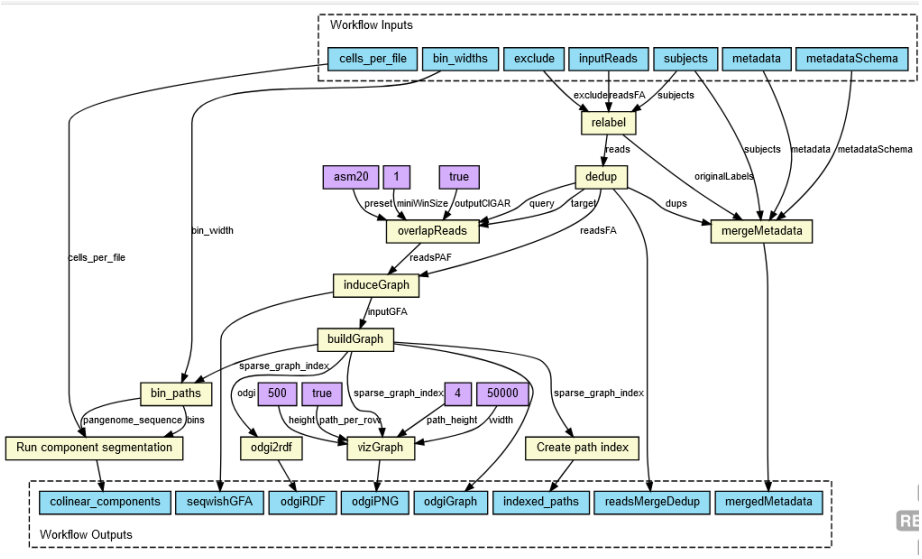

Figure 1: Workflow permettant d'identifier les variantes du SARS-CoV-2 et de suivre leur propagation.

1.2 Public visé

De nombreux cas d'usages dans divers domaines peuvent parvenir à l'utilisation de workflows :
- Les chercheurs l'utilisent pour partager le processus et les étapes expérimentales; cela permet d'utiliser des outils sans avoir de spécialiste en informatique.
- Le Data Scientist va l'utiliser pour analyser de grandes quantités de données facilement.
- Les chefs de projets en laboratoire peuvent l'utiliser pour faire une veille sur les outils existants et évaluer l'état actuel des connaissances.

Nous pouvons donc dire que le projet BIO4T a vocation a être utilisée par tout type de profil dans la communauté bioinformatique, tant par des personnes ayant des connaissances en informatique que par des personnes dont les connaissances dans ce domaine sont plus limitées.

1.3 Objectifs métiers

Cette section présente les objectifs métiers, les objectifs du projet, c'est-à-dire les besoins finaux des utilisateurs.

1.3.1 Objectif 1: Recherche de Workflows à partir du Langage Naturel

"En tant qu'utilisateur, je dois pouvoir entrer une description textuelle d'une tâche scientifique, et obtenir une liste de workflows d'opérations, accompagnée

de commandes spécifiques à exécuter."

1. Le système doit extraire des mots-clés EDAM (Ontologie) pertinents à partir de la description fournie par l'utilisateur.
2. L'utilisateur obtient les workflows associés construits par le système.
3. Les workflows trouvés doivent être classés par pertinence et affichés à l'utilisateur.

Les **Topics** et **Opérations** constituent des éléments clés permettant de structurer et d'orienter la recherche d'outils bioinformatiques en fonction des besoins de l'utilisateur. Ces concepts s'appuient sur l'ontologie **EDAM**, qui fournit une classification standardisée et hiérarchique des notions bioinformatiques.

1.4 Objectif 2 : Enrichissement Automatique des Workflows

Les outils sont associés à des mots clés d'opérations EDAM. Ces mots clés ont été définis par des humains et peuvent manquer de précision.

Il peut être nécéssaire d'enrichir ou reconstruire les mots clés liés aux outils.

Topics (Thématiques) : Les **Topics** représentent les domaines scientifiques ou thématiques associés à un outil ou à une recherche bioinformatique. Par exemple, des Topics tels que *Bioinformatics*, *Genomics* ou *Medical Informatics* permettent de catégoriser les outils selon leur champ d'application.

- **Rôle dans le projet :**
  - Faciliter le filtrage des outils en fonction du domaine scientifique.
  - Organiser les outils selon une classification standardisée issue d'EDAM.

Opérations : Les **Opérations** décrivent les actions ou processus spécifiques qu'un outil peut effectuer, comme *Sequence Alignment*, *Gene Annotation* ou *Data Visualization*. Ces opérations permettent de définir des workflows bioinformatiques complets.

- **Rôle dans le projet :**
  - Identifier les outils capables d'exécuter une opération précise demandée par l'utilisateur.
  - Construire des workflows combinant plusieurs opérations.

1.5 Etat de l'art, existant

1.5.1 Projet existant

Lorsque nous avons récupéré le projet, nous avons découvert que nous sommes la quatrième équipe à travailler dessus.

Durant l'année précédente, le groupe en charge du projet a contribué à la réécriture de la fonctionnalité de scoring des outils (figure 2). Pour cela, ils ont ajouté en propriété des outils un nombre, basé sur le format d'entrée et de sortie, qui indique quel outil est le plus adapté pour la fonctionnalité requise, en sortie d'un outil.

$$\frac{\sum_{0}^{n}(0.6 \times \text{score de confiance}) + (0.4 \times \text{score de co-occurrence})}{(2 \times n) - 1}$$

Figure 2: Formule du score d'un workflow

Ainsi, ici le score de confiance est un nombre entier, mis en attribut de chaque outil, permettant en fonction du nombre d'utilisation et les avis, de savoir si l'outil peut être utilisé sans soucis. Aussi, le score de co-occurrence est un indice permettant de déterminer si la sortie et l'entrée d'un outil est compatible. Cette modification permet la génération de workflows plus adaptés aux besoins des utilisateurs. Aussi, ils ont retravaillé l'Interface Humain-Machine, afin d'offrir une meilleure expérience aux utilisateurs, en facilitant la création de workflow. Ils ont également retravaillé les connexions et utilisations des API, afin d'optimiser la récupération des données. Enfin, ils ont pu mettre en place des améliorations au niveau de la base de données Neo4J, ainsi que l'insertion de données dedans, augmentant le nombre d'outils connus afin d'améliorer le fonctionnement de l'application. Pour finir, ils ont pu modifier la base NEO4J, afin de préparer la mise en place d'OpenSearch par notre groupe.

Jusqu'à présent, l'application disposait d'une interface, plutôt aboutie, qui permettait à l'aide d'un formulaire d'entrer des opérations, et affichait les workflows retournés (figure 3).

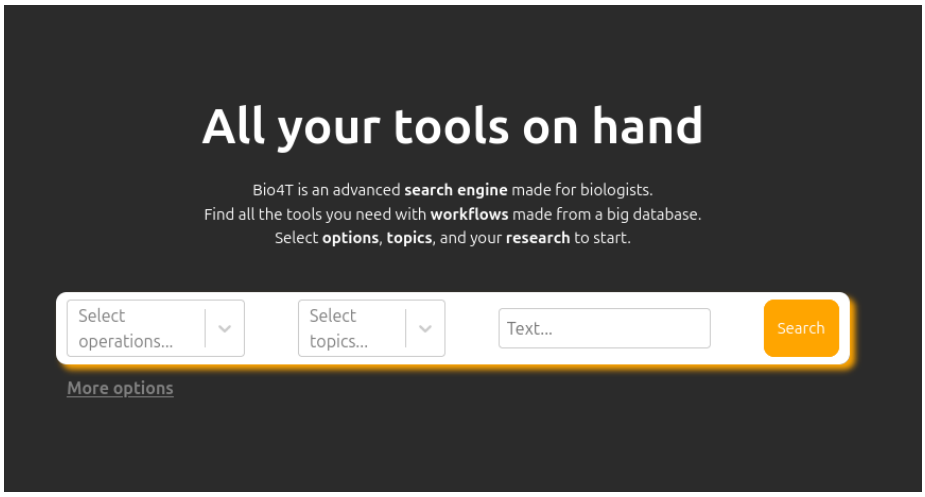

Figure 3: Interface du projet existant

Également, cette version de l'application permettait de récupérer des outils depuis biotools, ainsi que des opérations depuis Galaxy. Néanmoins, cela limitait les possibilités ainsi que la qualité des workflows retournés, car le nombre d'outils et d'opérations était limité.

1.6 Technologies utilisées

1.6.1 Biotools

Biotools est une plateforme recensant plus de 28000 outils bioinformatiques. L'entièreté des informations relatives à ces outils a été récupérée via l'API de biotools afin de peupler notre base de données.

1.6.2 Galaxy

Afin d'effectuer ceci, ils ont utilisé Galaxy, qui est une plateforme polyvalente de bioinformatique visant à rendre la science accessible aux chercheurs sans compétences spécifiques en programmation. La limitation de cette plateforme est qu'elle ne génère pas directement des workflows, mais elle permet d'en exécuter.

1.6.3 NEO4J

Egalement, pour le stockage de données, ils ont choisi une approche de base de données au format graphe, permettant de relier les outils ayant les mêmes opérations entre eux. Cette technologie de stockage de données est particulièrement adaptée pour le stockage de connaissance, car il existe beaucoup de liens entre les outils et les opérations. Pour ceci, ils ont fait le choix de NEO4J,

qui possède également une interface graphique (figure 4), afin de vérifier les requêtes et visualiser les interconnexions entre les données.

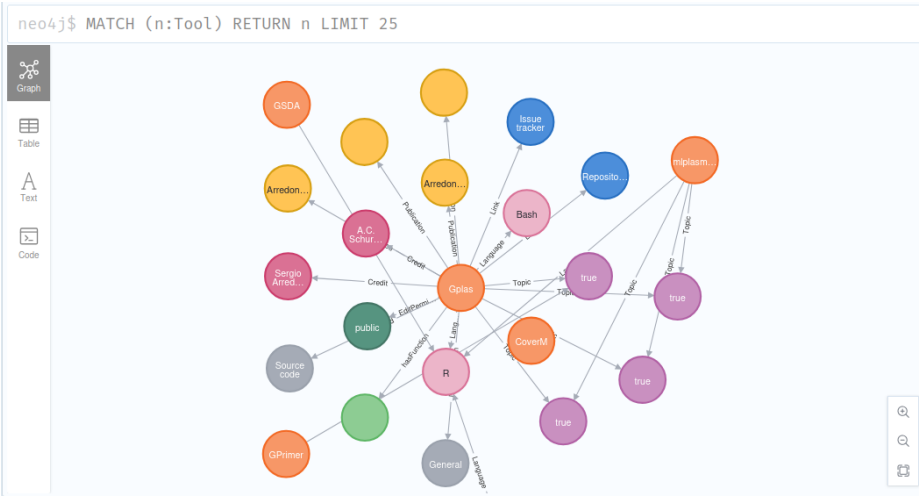

Figure 4: Interface de NEO4J, accessible sur navigateur.

1.6.4 Docker

Docker est un outil dit de conteneurisation, c'est à dire qu'il reprend des machines dans des états définis. Ces machines ont donc un système d'exploitation et des applications préinstallées dessus, leur permettatn d'être prêtes à être mise en service. Ce sont des conteneurs. En utilisant plusieurs conteneurs, nous pouvons simuler sur un même serveur ou ordinateur un réseau de plusieurs machines communicant entre elles, et cela nous permet d'avoir un conteneur par tâche spécifique, tout comme nous pourrions avoir plusieurs serveurs. Egalement, ce système permet d'augmenter rapidement la capacité d'un service si nécessaire, en démarrant une autre instance d'un conteneur. En effet, un conteneur est bien plus rapide à démarrer qu'une machine virtuelle.

2 Objectifs techniques

Ce chapitre définit les objectifs métier

2.1 LLM et Recherche de Workflows à partir du Langage Naturel

Le système doit analyser une requête en langage naturel et déduire les mots clés EDAM associés. Nous entrons dans le vaste domaine du NLP (Natural language processing), les principales difficultés sont :

> **Ambiguïté :** En langage naturel, **les mots sont uniques** mais peuvent avoir des **significations différentes** selon le contexte.
>
> **Synomie :** Un autre phénomène clé en langage naturel est le fait que nous pouvons **exprimer la même idée avec différents termes**

2.1.1 Technologie utilisée

Une approche très utilisée dans ce domaine est l'utilisation de LLM (Large Language Models), car ceux-ci sont spécialisés dans le traitement de texte. Ils sont également entraînés sur de nombreuses opérations sur du texte. Nous recherchons une méthodologie générale, afin de pouvoir les utiliser sur d'autres vocabulaires ou domaines. Cette solution permet d'avoir des résultats pertinents dans de nombreuses tâches de nature variées.

Nous ne nous occupons pas de la partie entraînement du modèle, mais seulement de l'utiliser, il n'est pas spécifiquement entraîné pour nos besoins. Dans notre cas, nous avons opté pour l'utilisation d'un modèle LLM de type BERT, en raison de l'existence de sa variante SCIBERT, spécialement entraînée sur des textes contenant des termes scientifiques. Ce choix nous permettra d'utiliser un modèle adapté au vocabulaire attendu, garantissant ainsi des résultats probants lors de la conversion du texte en mots-clés.

Néanmoins, l'utilisation de l'IA peut toujours porter à des erreurs, malgré l'utilisation de modèles robustes. Il n'est pas rare de voir des IA répondre de manière incohérente, appelées hallucinations. Dans notre cas, le modèle pourrait inventer des termes inexistants dans le vocabulaire EDAM. C'est pour cela que nous définissons une contrainte forte : **les termes en sortie du LLM doivent être inclus obligatoirement dans l'ontologie EDAM.** Cette règle permet de se protéger contre les hallucinations. De plus, les travaux des précédentes équipes par l'implémentation d'un score de confiance permettent d'assurer que le résultat final reste cohérent.

Nous allons utiliser OpenSearch pour utiliser le modèle, c'est une suite logicielle open-source dédiée à la recherche, à l'analyse de données et à la gestion des logs. Elle a été créée à partir du projet Elasticsearch et est soutenue par Amazon Web Services (AWS) et une communauté open-source. OpenSearch

permet de stocker, rechercher et analyser de grandes quantités de données de manière rapide et scalable.

Elle offre des fonctionnalités robustes pour la gestion d'index, l'analyse de texte et l'exécution de requêtes complexes, ce qui en fait une solution idéale pour notre projet. Nous pouvons ainsi effectuer des recherches dans des documents, à condition que ceux-ci soient indexés.

Pour cela, nous allons nous servir des différentes API (application programming interface) proposées par OpenSearch, afin de pouvoir interroger des modèles de machine learning. OpenSearch permet de faire une recherche sur un des données prédéfinies à partir de texte, comme du langage naturel. Il est possible d'utiliser plusieurs modèles de machine learning.

2.2 Objectif 2 : Enrichissement Automatique des Workflows

Afin d'obtenir des résultats plus pertinents, nous avons décidé d'augmenter le nombre de sources de données. En effet, jsuqu'à présent, les données provenaient essentiellement de biotools. Pour cela, nous avons donc récupéré les données depuis WorkflowHub, une plateforme en ligne permettant de référencer de nombreux workflows d'outils de tous domaines scientifiques. Dans ce but, nous nous connecterons donc à l'API de WorkflowHub afin de récupérer les workflow existants. Cela nous permettra d'avoir des informations pertinentes car ce sont des workflows déjà utilisés dans des conditions réelles, et donc qui sont pour la plupart reconnus par la communauté scientifique.

3 Description de la solution

3.1 Vers une base plus efficiente

3.1.1 Extraction des données

Pour rendre notre application plus cohérente, nous avons décidé de créer deux index. L'un est tool et servira pour les outils, l'autre est operation. Ces index ont été créés depuis les données Neo4J au lieu de celles de biotools. Ils seront ensuite utilisés avec OpenSearch pour générer nos résultats.

En plus de la création des deux index, nous avons décidé, comme expliqué en Section 2.2, d'intégrer les workflow existants provenant de WorkflowHub. Pour cela, nous avons donc réalisé plusieurs scripts Python. Le premier nous sert à créer des classes, nous permettant d'obtenir une représentation Python de worfkows. Aussi, un deuxième script permet de se connecter à l'API de WorkflowHub, afin d'en récupérer les worfklows sous format JSON. A l'aide du premier script Python, nous convertissons alors le worfkow en objet Python, afin de pouvoir réduire le nombre d'incohérences et d'erreurs résultant de données non homogènes. En effet, sur la quantité de données sur WorfklowHub, certains n'ont pas les mêmes champs que d'autres, des données sont manquantes, et certaines structures de données ne sont pas tout à fait identiques entre deux workflows.

3.1.2 Insertion de workflows préexistants

Une fois les classes Python créées, nous avons donc une certaine représentation des workflow, avec les données qui nous sont le plus utiles pour notre système.

.Ces données sont notamment des chaînes de caractères afin d'avoir une description du workflow, une liste d'étapes, contenant un outil et la liste des opérations effectuées, ainsi que pour chaque étape un pointeur vers la suivante. De plus, un système de poids a été mis en place, sous forme de nombre entier, afin de garder l'information des workflows les plus utilisés. Ce système de poids vient de WorkflowHub, et n'est pas mis à jour par notre système.

Ce fonctionnement nous a néanmoins posé un nombre élevé de problèmes, notamment en ce qui concerne le lien entre les outils présents dans les worfkow provenant de WorkflowHub et ceux déjà présents dans la base NEO4J.

Egalement, cette modification a entraîné des instabilités au démarrage, dans notre cas de l'environnement Docker. Ces problèmes étaient notamment liés à des erreurs dans les requêtes CYPHER vers NEO4J, que nous avons donc résolu au fur et à mesure.

Une fois que nous avons les données en Python, il nous restera alors à les insérer dans la base NEO4J. Pour ceci, nous avons donc utilisé le langage de requête CYPHER, puisque NEO4J se base dessus. Nous exécutons donc une requête afin d'intégrer les objets Python dans la base de données, en ayant au préalable créé la structure, par l'ajout de noeuds spécifiques dans NEO4J.

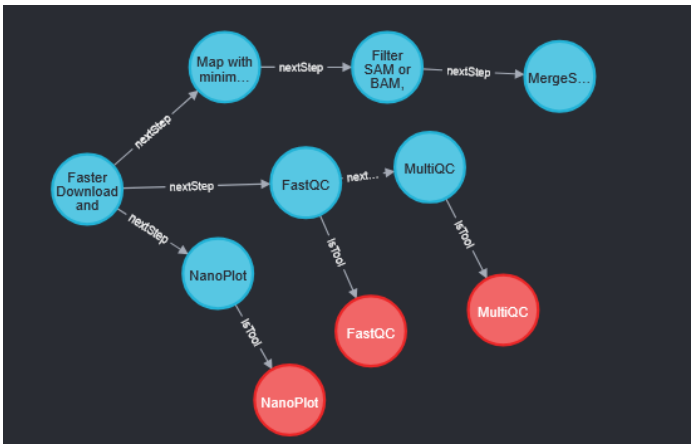

Figure 5: Exemple du workflow "Genomics - Read pre-processing" dans le navigateur Neo4J

3.1.3 Amélioration de la relation outil-opérations

Cette nouvelle version de la base de données nous a permis d'augmenter le nombre de nouveaux workflow possibles, que nous présentons à l'utilisateur. Néanmoins, un problème subsiste : Le lien entre outils et opérations ne permet pas d'utiliser pleinement l'ontologie EDAM. En effet, jusqu'à présent les outils sont bien reliés à des opérations, mais celles-ci ne sont pas assez spécifiques, et la création de nouveaux workflows en est donc impactée du fait que les .

Notre objectif est donc d'améliorer le lien entre outils et opérations, car le fonctionnement de l'application est de partir des opérations pour retrouver les outils. Pour cela, nous allons donc prendre les descriptions des outils, lorsqu'elles sont présentes sur les plateformes telles que biotools, et créer une requête OpenSearch. Cette requête permet, via le LLM, de les comparer aux termes de l'ontologie EDAM et donc de pouvoir avoir une meilleure classification des outils.

Grâce à ceci, nous espérons donc obtenir des résultats plus pertinents, mais surtout un plus grand nombre de résultats. Jusqu'à présent, si l'utilisateur entrait une requête trop précise, il était courant que BIO4T ne renvoie simplement aucun résultat, car avec le système précédent de lien entre outils et opérations, celles-ci étaient trop éloignées de la demande de l'utilisateur.

3.2 Moteur de recherche OpenSearch

3.2.1 OpenSearch

OpenSearch est un moteur de recherche basé sur des index, donc des critères de stockage de données, permettant une certaine approximation, pour renvoyer plusieurs résultats similaires par requête. Dans le cadre de l'intégration

d'OpenSearch à notre projet, nous avons principalement travaillé sur la construction et l'indexation de deux bases essentielles à la recherche sémantique : les opérations issues de l'ontologie EDAM et les outils de la base de donnée existante.

3.2.2 Indexation des opérations EDAM

Cette indexation permet de recréer les index avec les changements futurs en base de donnée, par exemple pour prendre en compte l'ajout d'outils ou d'opérations.

À partir du fichier `EDAM.owl`, nous avons développé un script (`IndexOperationsManager.py`) pour extraire les opérations, leurs définitions, ainsi que les relations hiérarchiques parent-enfant. Cette extraction s'est faite via des requêtes SPARQL sur un graphe RDF, en utilisant la bibliothèque `RDFLib`. Pour chaque opération, un vecteur de similarité sémantique a été généré à l'aide du modèle `SciBERT` (pré-entraîné sur des articles scientifiques) via la librairie `transformers`. Ces vecteurs représentent le sens des définitions et permettent une recherche vectorielle *K-NN* dans OpenSearch. Le résultat a été exporté dans un fichier `doc_operations.json`, prêt à être indexé.

3.2.3 Indexation des outils existants

En parallèle, nous avons également conçu un second script (`IndexToolsManager.py`) permettant de récupérer automatiquement les outils depuis l'API publique de `bio.tools`. Pour chaque outil, la description textuelle et la liste des opérations associées ont été extraites. Comme pour les opérations, un vecteur sémantique a été calculé via `SciBERT`. L'ensemble des documents a été sauvegardé dans le fichier `doc_outils.json`.

Les données sont donc structurées dans l'espace vectoriel de recherche. Dans notre cas pour chaque opération on obtient un vecteur dense de 768 valeurs, comprises entre -1 et 1, ce qui donne quelque chose comme suit pour chaque mot clé :

```
{"titre": "Recombination detection", "contenu": 'Detect
    recombination (hotspots and coldspots) and identify
    recombination breakpoints in a sequence alignment.",
    "vector": [-0.008221910335123539, -0.051848314702510834,
    -0.3008790612220764, -0.11104746907949448,
    0.17261949181556702, -0.5275784730911255...
}
```

Plus les embeddings sont précis, plus ils sont capables de capturer des nuances sémantiques. Cela permet de retrouver une opération similaire même si les mots exacts ne sont pas répertoriés dans la requête, permettant une recherche sémantique avec l'implémentation de la fonction search_semantically.

3.2.4 Perspectives

Ces deux index jouent un rôle central dans la recherche sémantique par OpenSearch. Ils permettent notamment de :

- proposer dynamiquement des opérations à partir d'une description en langage naturel ;
- associer un outil à une ou plusieurs opérations, même si ces dernières ne sont pas explicitement renseignées dans les métadonnées de `bio.tools` ;
- alimenter le moteur de suggestion et de filtrage des outils dans la nouvelle interface de recherche.

Le couplage entre les descripteurs vectoriels issus de `SciBERT` et OpenSearch offre de nombreuses perspectives : amélioration des suggestions automatiques, détection de similarités implicites entre outils, ou encore validation automatique des annotations EDAM sur les outils.

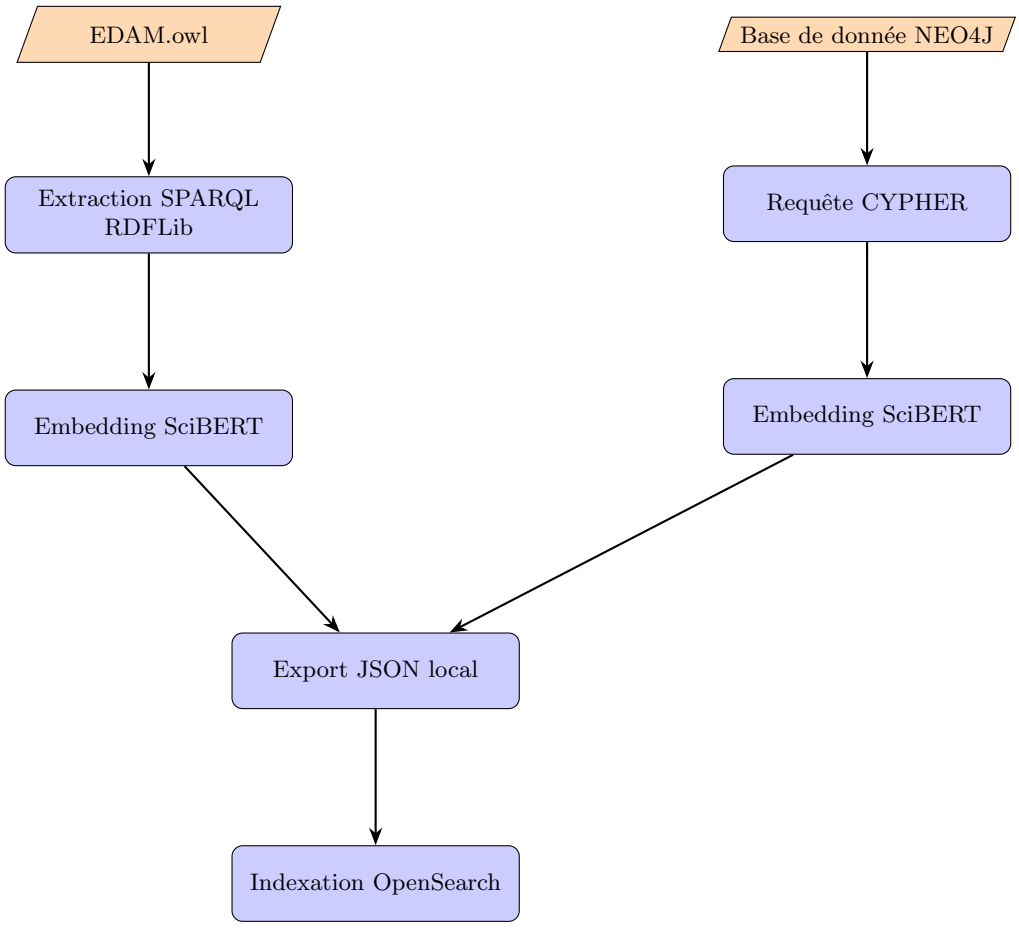

Figure 6: Pipeline d'indexation des opérations et des outils dans OpenSearch

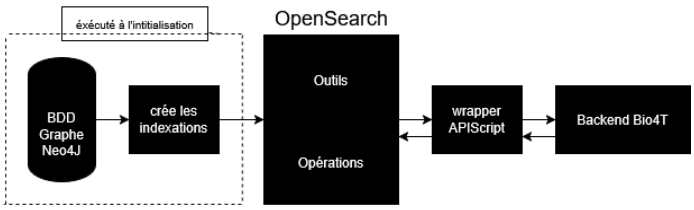

Figure 7: Interactions avec OpenSearch

3.3 Intégration de la recherche sémantique

3.3.1 En Back-End

On utilise le wrapper APIScript pour appeller la fonction search_semantically qui permet à partir d'un texte de retourner une liste de tuple (nomOperation, score), le score est compris entre 0 et 1

A partir de cela, on interface avec le système existant.

```
def request(self, textQuery: str, topics: List[str]) ->
    List[Dict[str, List[Dict[str, str]]]]:
    # Get operations and topics with the textQuery
    matches = search_semantically(textQuery, 'operations',
    'titre', 5)

    print("========LLM SEARCH OPERATIONS==========",
    file=sys.stderr)
    operations = [operation for operation, _ in matches]
    for operation, score in matches:
        print(f"Operation: {operation}, Score: {score}")

    (opWorkflows, wfs_operations) =
    self.weightedWorkflows(operations)
```

Ainsi on modifie la route API d'envoi du formulaire en replacent la liste d'opérations par un champ 'text' dans le JSON.

```
@app.route("/sendForm", methods=['POST'])
    def getWorkflow() -> str:json_dict = request.get_json()
    result = request_handler.request(json_dict["text"],
    json_dict["topics"])
```

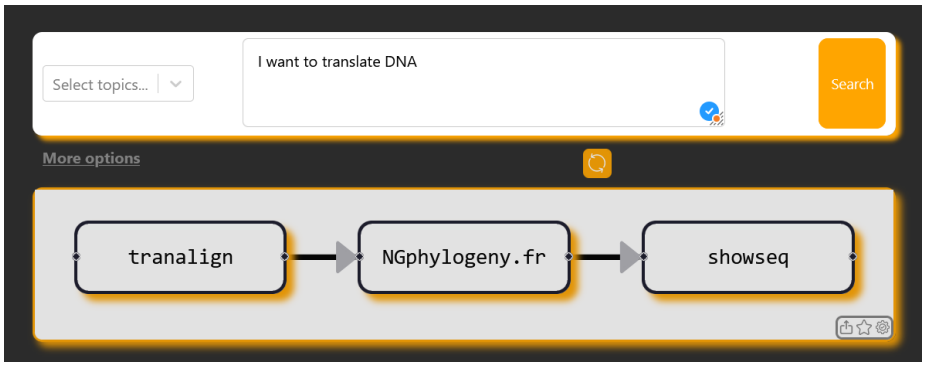

Figure 8: Interface Web modifiée

3.3.2 Front-End

Nous avons ajouté une page V3 avec le nouveau formulaire pour permettre l'ajout de texte. Le champ Topic optionnel, sert à orienter les résultats.

3.4 Approche expérimentale

À partir de cette étape, nous avons pu tester notre application de bout en bout. Dans un premier temps, l'application ne retourne aucun workflow, la liste retournée est vide. Ainsi, il nous a fallu déboguer et trouver quelles sont les raisons qui nous mènent à ce résultat :

- Dans un premier temps, la fonction **search_semantically** renvoie bien les opérations liées au texte d'entrée. Elles sont suffisamment précises et correspondent bien à la requête.

```
127.0.0.1 - - [02/May/2025 15:56:15] "OPTIONS /sendForm HTTP/1.1" 200 -
========LLM SEARCH OPERATIONS===========
Operation: DNA translation, Score: 1.7103405
Operation: DNA back-translation, Score: 1.6908895
Operation: Sequence conversion, Score: 1.6851238
Operation: DNA linear map rendering, Score: 1.6848643
Operation: Data retrieval (gene annotation), Score: 1.6771998
```

Figure 9: Résultat pour la requête : "I want to translate DNA"

- Ensuite, la fonction **weightedWorkflows** est appelée, elle recherche dans les exemples de workflows, les opérations similaires. Elle calcule un score de compatibilité pour chaque exemple de workflow à partir des opérations. Ensuite, un filtre est appliqué avec un seuil requis. Afin d'assurer d'avoir

des enchainements cohérents d'outils. À cette étape, aucun workflow n'a un score suffisamment élevé, nous en avons déduit que le problème se situait dans cette fonction.

- Pour savoir pourquoi aucun workflow est valide, il faut s'intéresser plus précisément à l'algorithme utilisé : elle prend en paramètre une liste d'opérations. Pour chaque workflow d'exemple, si une opération en paramètre correspond à une opération du workflow, le score augmente (figure 2).

- Hors, lors du débogage, nous avons constaté que les opérations en paramètres ne sont pas présentes dans les workflows modèles. À cette étape, nous nous sommes réunis pour trouver comment résoudre ce problème et trouver des solutions :

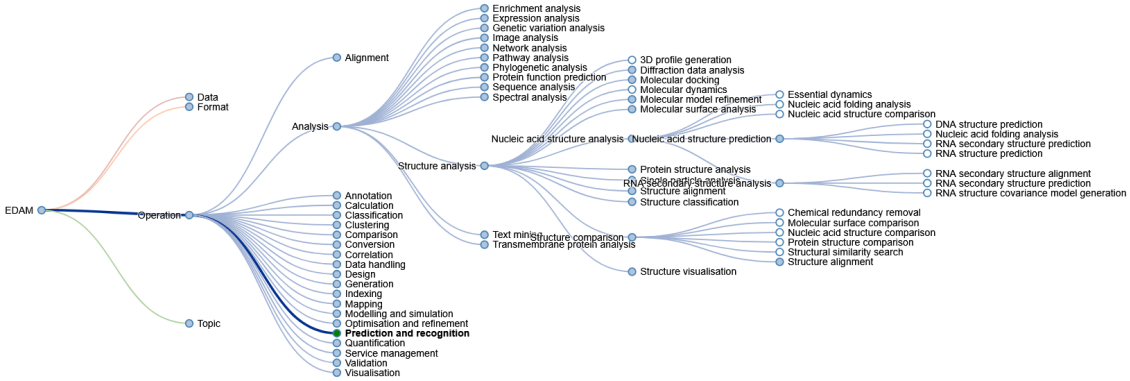

Figure 10: exemple d'arborescence EDAM

3.4.1 Voies d'améliorations étudiées

Dans un premier temps, nous avons envisagé l'instauration d'un lien de parenté entre les différentes opérations, de manière à ajuster le score attribué en fonction de leur proximité ou de leur similarité. Cette réflexion s'appuie sur un indice laissé par l'équipe précédente : le système permet déjà un calcul de score pondéré selon les opérations. Toutefois, nous avons constaté que, dans l'implémentation actuelle, tous les poids sont fixés à 1. Cela soulève une interrogation : pourquoi avoir mis en place un mécanisme de pondération, si aucune différenciation n'est actuellement exploitée ? Cette incohérence suggère une intention initiale non aboutie, que nous proposons de reprendre et d'approfondir dans notre démarche.

La première solution envisagée est donc d'attribuer un poids selon le niveau de parenté entre les deux opérations, en s'appuyant sur les fonctionnalités apportées par la base de donnée orientée graphe NEO4J.

De plus, nous avons remarqué, lors de la comparaison des résultats obtenus avec le LLM et opérations présentes sur les workflows d'exemples, les problèmes suivants :

- Le modèle LLM renvoie une plus grande diversité d'opérations, plus qualitatives qui sont le plus souvent des feuilles dans le graphe EDAM.
- Les opérations annotées sur les workflows sont peu diversifiés et de très faible précision, sur des noeuds en haut du graphe EDAM. Dans l'immense majorité des cas, on pourrait déduire des mots clés plus précis à partir de la description des outils. On revient à notre second objectif : enrichir le lien entre les workflows et les opérations.

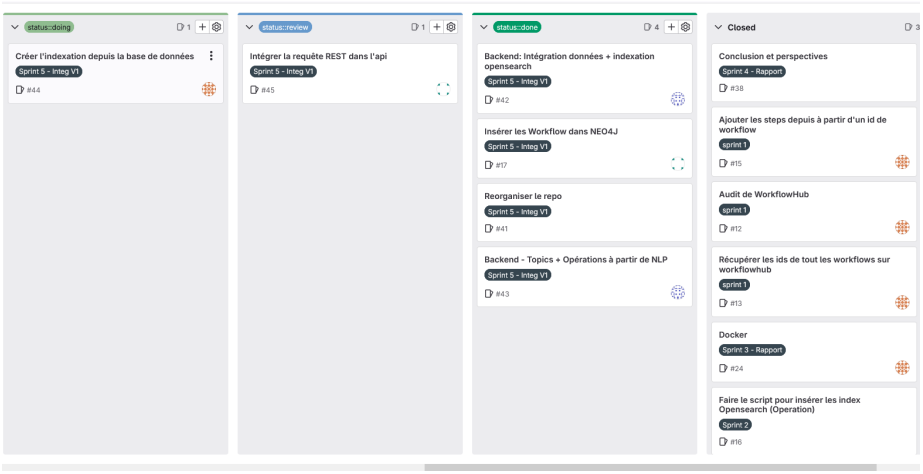

Figure 11: Vue de la fonctionnalité Issue boards de GitLab

4 Gestion de projet

4.1 Répartition des tâches

Pour travailler sur nos taches, nous étions en groupe de deux ou trois, pour éviter de rester bloquer sur une difficulté. Nous avons utilisé la fonctionnalité Issue boards de GitLab pour voir l'avancement dans le projet. Celle-ci n'est pas réellement faite pour cela, mais l'utiliser nous évitait d'avoir besoin d'un autre outil externe tel que Trello pour avoir une vision globale des tâches et des membres du groupe associés. Egalement, nous avions un point par semaine où nous nous réunissions sans l'encadrant de projet, ainsi qu'un autre point avec lui. Grâce à ces points, nous avons pu utiliser une méthode agile, ressemblant à Scrum. Ces points ont permis de faire part de nos difficultés et de trouver des solutions adaptées.

4.2 Impact des obstacles techniques sur la planification

Un temps important a été consacrée à l'obtention de la version initiale, au vu du nombre d'étapes requises. De plus, nous avons surmonté plusieurs difficultés majeures :

- **Les aspects DevOps ont représenté un point de ralentissement** : l'environnement de développement et de déploiement manquait de flexibilité et demandait beaucoup de temps à être déployé. Les nombreuses initialisations requises (Base de données, scoring, indexation) ont fait perdre un temps précieux.
- **Les effets de bord** liés à l'absence de parentalité entre les opérations, ainsi les données anciennes sont devenues obsolètes.

- **L'adaptabilité des données s'est imposée comme un enjeu central**. Le système doit être capable de s'ajuster aux évolutions de la base de donnée.

La conséquence est qu'un nombre important de tâches n'étaient pas prévues initialement. Ce qui est la cause de retards sur les fonctionnalités attendues, des efforts nécessaires pour avoir des résultats pertinents ou assurer la continuité du projet.

Ainsi, nous avons obtenu une version satisfaisant les objectifs début avril.

5 Conclusion

Notre itération s'est inscrite dans une démarche à la fois exploratoire et applicative visant à développer une recherche à partir de langage naturel. Pour ce faire, nous avons utilisé un modèle d'intelligence artificielle LLM. Nous avons pu concevoir et mettre en œuvre une solution en s'appuyant sur des techniques et technologies récentes, dans un environnement comportant de nombreuses technologies dans le domaine spécifique de la bioinformatique.

La participation à ce projet nous a permis d'identifier les difficultés concrètes liées à la mise en œuvre de ce type de solution : contraintes DevOps, gestion de l'évolution des données, mise en place de nouvelles technologies.

Cette expérience nous a été formatrice en gestion de projet, ainsi que sur les contraintes d'organisation et de conception. Ainsi, cette étape nous a permis d'implémenter un moteur de recherche sémantique.

Nous avons pu améliorer la qualité de nos données en s'appuyant sur de nouvelles sources, telles que workflowhub, tout en ouvrant la voie à de nouvelles améliorations à venir. Par exemple, l'utilisation de Weaviate pourrait être une bonne alternative à OpenSearch, par sa supériorité lors de la création de vecteurs depuis des index, et également le fait qu'il utilise des modèles de LLM plus modernes.

Nous remercions notre encadrant, François Moreews pour nous avoir suivi et conseillé cette année.