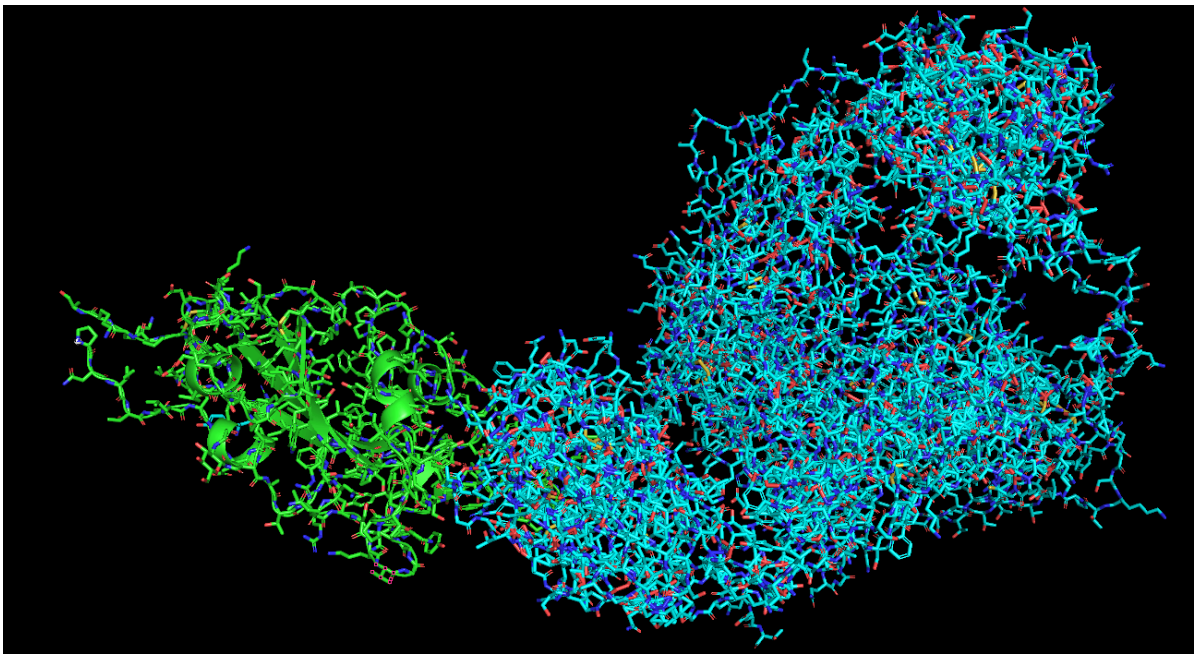
Projet Recherche Innovations

Bio-informatique : Sélection d’anticorps



Antoine Delenclos
22210070

Sommaire

Delenclos Antoine

Tuteurs : M.Bocquillon
M.Néron

Remise en contexte et cahier des charges

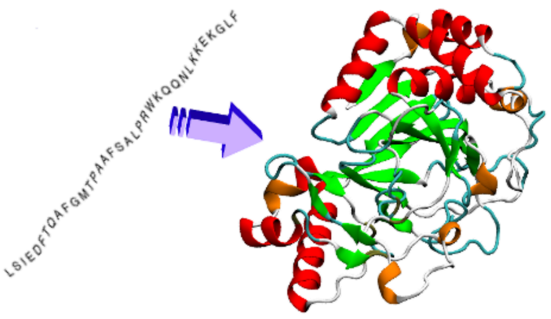
Remise en contexte :

Ce projet a été réalisé de septembre à février dans le cadre du Projet Recherche & Innovation.
Les tuteurs de ce projet sont M.Bocquillon et M.Néron.
Le projet vise à développer une librairie d'anticorps humains capables de neutraliser *Streptococcus agalactiae*, une bactérie responsable d'infections néonatales graves. Ce pathogène est à l'origine de complications sérieuses chez les nouveau-nés, avec un taux de mortalité de 10 % et des séquelles fréquentes chez les survivants. L'émergence de souches multirésistantes et les perturbations du microbiote vaginal causées par les antibiotiques renforcent la nécessité de trouver des alternatives thérapeutiques. Ce projet s'inscrit dans une collaboration avec le laboratoire UMR 1282 ISP et vise à utiliser des outils bio-informatiques pour sélectionner les meilleurs candidats anticorps.
Ne pouvant pas directement tester sur *Streptococcus agalactiae,* nous avons testé notre pipeline sur la protéine Spike du SARS-CoV-2 Wuhan (variant Omicron), car nous avons déjà des fichiers de résultats obtenus en laboratoire.

Un peu de définition pour bien comprendre en quoi consiste le projet :

Une protéine est une macromolécule biologique essentielle à la vie, composée d'une chaîne d'acides aminés liés entre eux.

Première lettre \ Seconde lettre	U	C	A	G	Troisième lettre
U	UUU, UUC : Phe UUA, UUG : Leu	UCU, UCC, UCA, UCG : Ser	UAU, UAC : Tyr **UAA Stop** **UAG Stop**	UGU, UGC : Cys **UGA Stop** UGG Trp	U C A G
C	CUU, CUC, CUA, CUG : Leu	CCU, CCC, CCA, CCG : Pro	CAU, CAC : His CAA, CAG : Gln	CGU, CGC, CGA, CGG : Arg	U C A G
A	AUU, AUC, AUA : Ile AUG Met	ACU, ACC, ACA, ACG : Thr	AAU, AAC : Asn AAA, AAG : Lys	AGU, AGC : Ser AGA, AGG : Arg	U C A G
G	GUU, GUC, GUA, GUG : Val	GCU, GCC, GCA, GCG : Ala	GAU, GAC : Asp GAA, GAG : Glu	GGU, GGC, GGA, GGG : Gly	U C A G



Sa structure :

- Séquence primaire : Ordre des acides aminés (ex: Ala-Leu-Val-...), déterminé par l'ADN via des triplets de nucléotides (A, U, C, G).
- Repliement 3D : La chaîne se plie en structures complexes (hélices α, feuillets β) pour former une forme fonctionnelle. C'est ce qu'on appelle la structure quaternaire.

Ses fonctions :

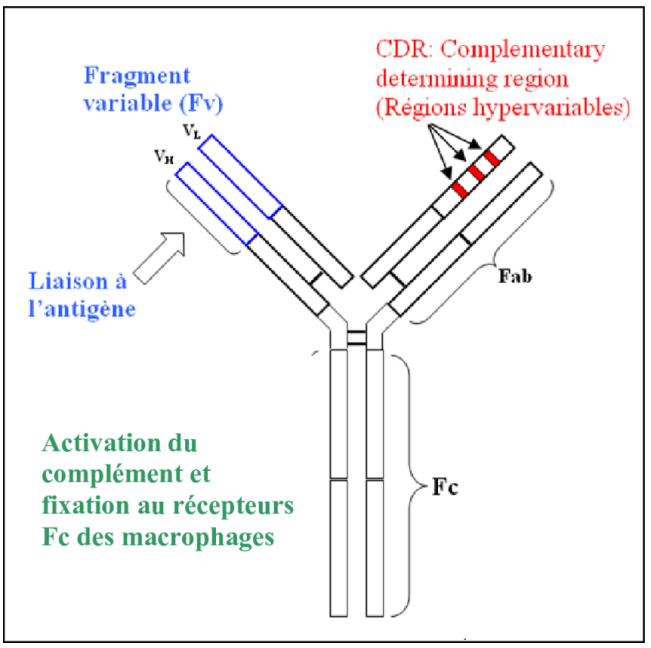
- Catalyse (enzymes)
- Défense (anticorps)
- Transport (hémoglobine)
- Structure (collagène)

Par exemple, la protéine Spike du SARS-CoV-2 8DV1 permet au virus de se fixer aux cellules humaines. Cette protéine spike est la protéine cible qu'on essaiera de neutraliser dans ce projet.

Les immunoglobulines sont des protéines spécialisées produites par les lymphocytes B pour neutraliser des agents pathogènes (bactéries, virus).

Sa Structure :

- Fragment Fab :
    - Régions FWR (Framework) : Structure stable, conservée entre les anticorps.
    - Régions CDR (Complementarity Determining Regions) : Zones hypervariables formant le site de liaison à l'antigène (ex: FbsA de Streptococcus).
- Fragment Fc :
    - Active le système immunitaire (complément, macrophages).



Dans notre sujet nous nous intéresserons uniquement à la partie Fab car c'est la partie qui va se lier à l'antigène, et qui a donc l'effet neutralisant s'il existe.

Fonctions clés d'une immunoglobuline :

- Neutralisation : Blocage de l'antigène (ex: empêcher S. agalactiae d'adhérer aux cellules).
- Opsonisation : Marquage des pathogènes pour les phagocytes.
- Activation du complément : Destruction des membranes bactériennes.

On cherche à obtenir des biomédicaments anti-infectieux, c'est-à-dire des immunoglobulines humaines qui vont être reproduites car elles ont un effet neutralisant sur la protéine cible.

Cahier des charges :

Le but de ce projet était d'avoir une pipeline prenant en entrée un fichier CSV et donnant en sortie la liste de certains immunoglobulines susceptibles de neutraliser le pathogène cible.

Il fallait également fournir des guides permettant d'expliquer l'installation des outils afin que le travail puisse être repris par la suite.

Etat de l'art

La reconstruction des séquences d'anticorps à partir de données de séquençage haut débit (CSV de clonotypes) est un défi central en immunologie computationnelle. Les outils comme IMGT/HighV-QUEST ou MiXCR existent mais notre approche vise à mettre en place une pipeline open-source pour que le laboratoire UMR 1282 ISP, et d'autres par la suite, puisse l'utiliser sans problème.
Lors de la recherche d'outils/API pour réaliser ce projet, j'ai fait de nombreuses recherches et j'ai trouvé AlphaFold qui est un outil développé par Google DeepMind, la branche IA de Google, qui permet de reconstruire la forme quaternaire d'une protéine à partir de sa séquence d'acides aminés. Un autre outil qu'ils ont développé est AlphaProteo qui permet de donner une protéine cible et ensuite le modèle d'IA va générer une structure quaternaire qui devrait être neutralisante. Ce dernier outil, comme AlphaFold 3, n'a pas encore été mis à disposition du grand public malheureusement.

MAbSilico, une entreprise locale, fait un travail un peu similaire. Ils travaillent avec différents laboratoires à travers le monde, et ont pour but d'accélérer, via des méthodes d'intelligence artificielle, le développement d'anticorps (21 jours contre plusieurs mois en laboratoire) pour une cible précise.

Pipeline

Le projet consistait à obtenir une pipeline complète et de la tester sur des fichiers CSV.

Pour le test la protéine cible choisie a été Spike SARS-CoV-2 Wuhan (8DV1) car nous avons les résultats de sélection d'immunoglobulines pour ce pathogène.

La pipeline consiste donc à prendre en entrée un fichier CSV, de réussir à en extraire les différentes séquences de l'immunoglobuline, de prédire la structure quaternaire des chaînes, d'assembler l'immunoglobuline, de faire du docking moléculaire et d'obtenir les énergies de liaison de ce docking.

La logique derrière cette méthode est qu'après docking avec la région cible, plus les énergies de liaison seront faibles, plus les liaisons seront stables et on peut s'attendre à ce que cette immunoglobuline s'accroche bien au pathogène.

CSV et reconstruction de séquences

Le projet consistait à obtenir une pipeline complète et de la tester sur des fichiers CSV.

Pour le test la protéine cible choisie a été Spike SARS-CoV-2 Wuhan (8DV1) car nous avons les résultats de sélection d'immunoglobulines pour ce pathogène.

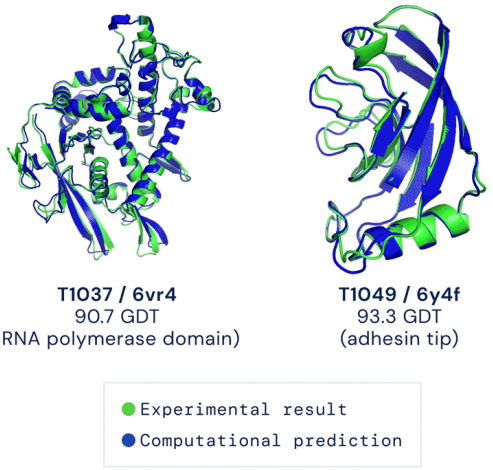
La pipeline consiste donc à prendre en entrée un fichier CSV, de réussir à en extraire les informations nécessaires pour reconstruire chaque chaîne. En effet le fichier CSV contient 2 lignes par patient : une pour la chaîne lourde et l'autre pour la chaîne légère. Une immunoglobuline est composée de 2 chaînes lourdes identiques et de 2 chaînes légères identiques. Comme dit précédemment, on ne s'intéresse qu'à la zone Fab de chacune des chaînes, cela tombe bien car on a les séquences d'acides aminés des FWR et CDR. On peut donc reconstruire la séquence complète en faisant FWR_1 + CDR_1 + FWR_2 + CDR_2 + FWR_3 + CDR_3 + FWR_4.

J'ai également fait en sorte de pouvoir le faire via les gènes V,D,J,C en créant des scripts qui vont chercher sur https://www.ensembl.org/index.html ou https://www.uniprot.org/ les gènes correspondant. Mais dans notre cas c'est peu pertinent car la séparation des régions à partir des gènes a déjà été faite par le laboratoire.

Prédiction de la structure quaternaire avec AlphaFold 2

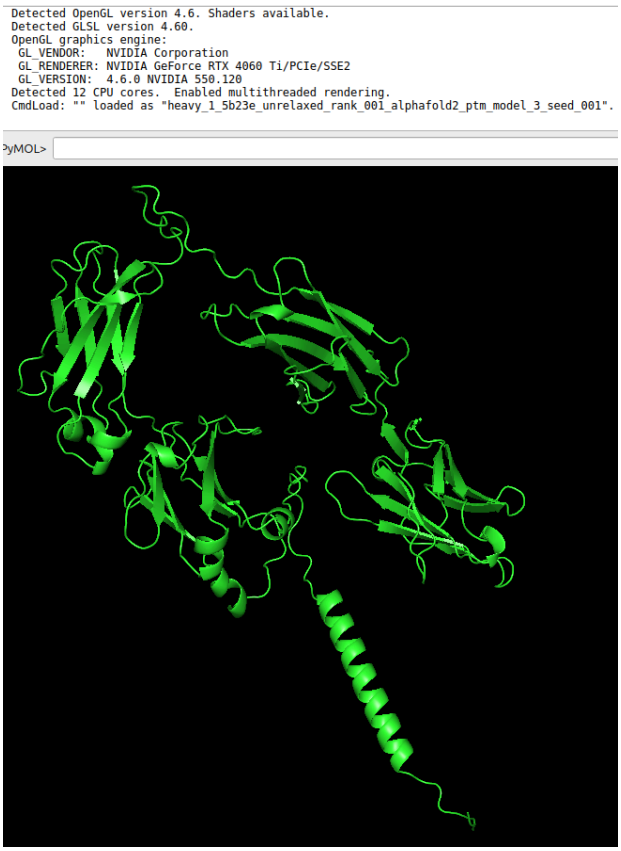
Une fois ces chaînes obtenues, j'utilise le modèle “monomer” d’AlphaFold 2 qui va permettre d’obtenir une prédiction de la structure quaternaire de la protéine à partir de sa séquence d’acides aminés. Il existe également le modèle “multimer” qui peut prendre plusieurs séquences dans un fichier FASTA mais le problème est que ce modèle n’est fiable qu’avec AlphaFold 3, qui n’est pas disponible.

Pour ces prédictions le temps de calcul est d’environ 1 heure pour la chaîne lourde et de 10 minutes pour la chaîne légère. Le temps de calcul augmente exponentiellement avec la longueur de la séquence d’acides aminés. J’ai laissé tous les paramètres par défaut car je n’ai pas eu le temps d’expérimenter sur plusieurs paramètres, mais les résultats semblent concluant avec les paramètres par défaut malgré tout.



Alignements de chaînes et assemblage de l'immunoglobuline

Une fois le traitement par AlphaFold fait, il faut un moyen d'ouvrir ces fichiers, au format PDB, et d'assembler l'immunoglobuline complète.
Pour la visualisation, j'utilise PyMol qui permet de visualiser la protéine mais qui va également permettre via sa bibliothèque Python, d'utiliser les commandes dans des fonctions Python, par exemple pour sélectionner certains acides aminés spécifiques.



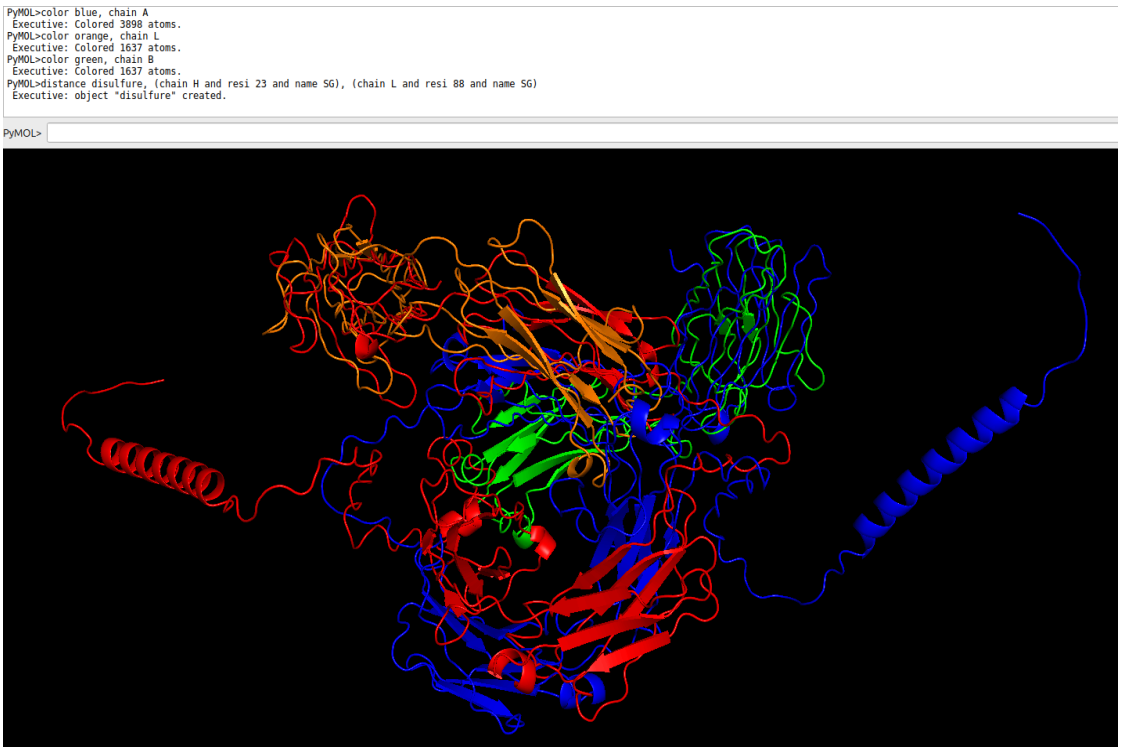
L'alignement des chaînes lourdes (IGH) et légères (IGK/IGL) est une étape critique pour reconstituer le fragment variable de l'immunoglobuline, responsable de la liaison à l'antigène. À partir des prédictions individuelles générées par AlphaFold, les chaînes sont traitées séparément. Pour les aligner, on sélectionne spécifiquement les régions variables (environ 110 premiers résidus) en se basant sur les atomes Carbone-α (squelette peptidique), garantissant une superposition précise des structures. Le module Superimposer de BioPython est utilisé pour minimiser l'écart quadratique moyen (RMSD) entre les deux chaînes, selon la formule :

$$RMSD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\delta_i^2}$$

```python
def alignChains(heavy_pdb, light_pdb, output_pdb):  2 usages
    #Rename chains ID
    modifyChainsId(heavy_pdb, heavyBool: True)
    modifyChainsId(light_pdb, heavyBool: False)

    # Configuration initiale
    parser = PDBParser(QUIET=True)
    io = PDBIO()

    # Charger et renommer les chaînes
    heavy = parser.get_structure("heavy", "renamed_heavy_chain.pdb")
    for chain in heavy[0]:
        chain.id = "H"  # Force la chaîne lourde à être "H"

    light = parser.get_structure("light", "renamed_light_chain.pdb")
    for chain in light[0]:
        chain.id = "L"  # Force la chaîne légère à être "L"

    # Aligner les domaines variables (VH sur VL) généralement avant 110.
    vh_atoms = [atom for res in heavy[0]["H"] if res.id[1] <= 110 for atom in res if atom.name == "CA"]
    vl_atoms = [atom for res in light[0]["L"] if res.id[1] <= 110 for atom in res if atom.name == "CA"]

    sup = Superimposer()
    sup.set_atoms(vh_atoms, vl_atoms)  # Aligner VL sur VH
    sup.apply(light[0]["L"].get_atoms())  # Appliquer la transformation à la chaîne légère

    # Créer une structure combinée
    combined = StructureBuilder.Structure("antibody")
    model = Model.Model(0)
    model.add(heavy[0]["H"].copy())  # Ajouter la chaîne lourde
```

On va donc faire cet alignement pour le couple chaîne lourde/chaîne légère puis dupliquer cette nouvelle structure et faire un alignement entre ces 2 clones de couples.

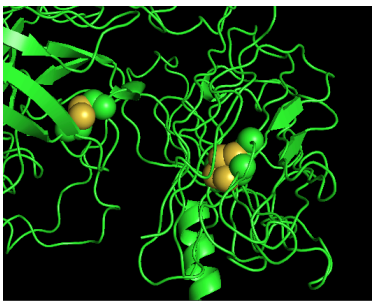
Une fois cela fait on aura donc la structure quaternaire complète de notre immunoglobuline.


PyMOL>color blue, chain A
Executive: Colored 3898 atoms.
PyMOL>color orange, chain L
Executive: Colored 1637 atoms.
PyMOL>color green, chain B
Executive: Colored 1637 atoms.
PyMOL>distance disulfure, (chain H and resi 23 and name SG), (chain L and resi 88 and name SG)
Executive: object "disulfure" created.
PyMOL>

Vérification de la structure

Vu que j'ai utilisé le mode "monomer" d'AlphaFold il faut vérifier si la structure de notre immunoglobuline est correcte, ou du moins logique. En effet, malgré notre alignement, il est possible que notre immunoglobuline ne corresponde pas à quelque chose d'existant dans la vraie vie. Pour cela je fais la vérification des ponts disulfure. Les ponts disulfure sont des structures qui existent dans toute immunoglobuline entre les chaînes lourdes et chaînes légères. Si on n'en trouve pas dans notre structure d'immunoglobuline on peut donc l'invalider. Ils correspondent à des liaisons covalent qui se forme entre les atomes de soufre de 2 cystéines et dans une distance entre [2.0 Å, 2.5 Å].

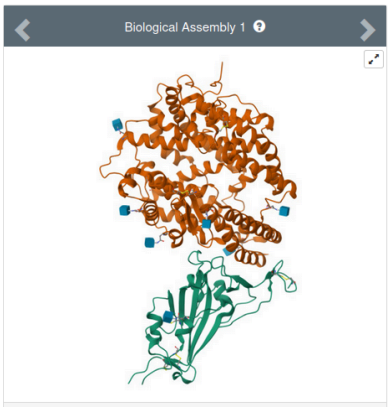

```
=== Ponts disulfures détectés ===
A-98 ↔ B-88
A-264 ↔ A-321
A-367 ↔ A-430
B-134 ↔ B-194
H-98 ↔ L-88
H-264 ↔ H-321
H-367 ↔ H-430
L-134 ↔ L-194
```

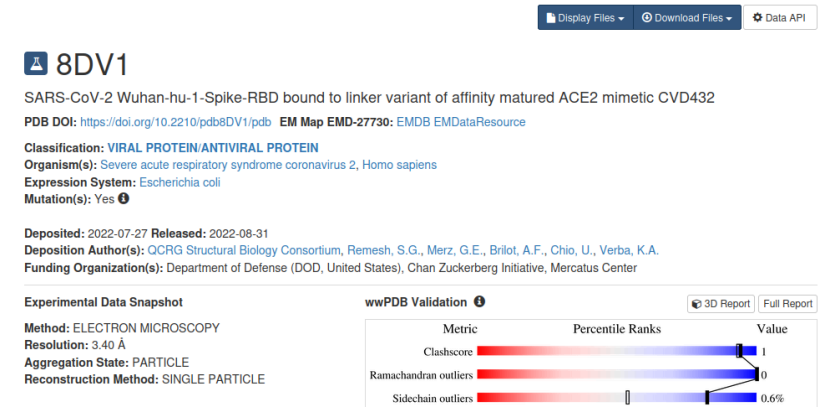


Pour chaque test que j'ai effectué j'ai retrouvé ces structures, j'en ai donc déduit que la reconstruction de la structure quaternaire d'une immunoglobuline à partir des séquences d'acides aminés devrait fonctionner.

Obtention de la protéine cible

Une fois la structure quaternaire de l'immunoglobuline obtenue, encore fallait-il obtenir la structure quaternaire de la protéine Spike voulue. Pour ce faire j'ai décidé de créer un script qui va récupérer sur UniprotKB la protéine 8DV1 qui contient la structure de la protéine Spike et la structure d'une immunoglobuline associée. Ensuite je compare les structures et je garde celle qui est la plus petite.

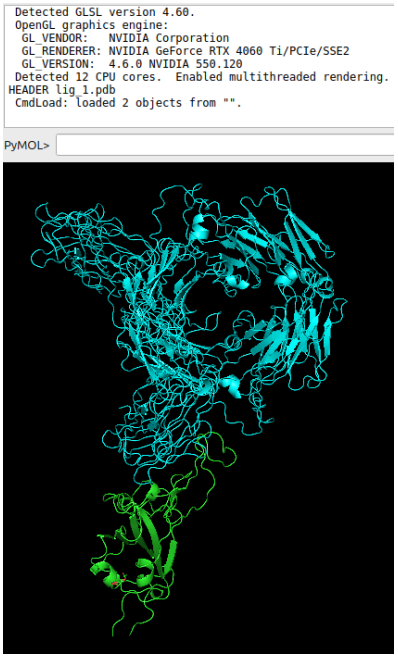




Docking moléculaire et énergies de liaison

Enfin, nous avons tous les prérequis nécessaires afin de faire le docking moléculaire. Le but du docking moléculaire va être de trouver des sites d'accroche et de calculer les énergies de liaison de ces sites afin de déterminer leur stabilité (plus l'énergie de liaison est faible plus la liaison est stable). Pour ce faire j'ai utilisé AutoDock Vina et HDOCK (seul HDOCK m'a donné des résultats donc c'est le seul que je mentionnerai).

Exemple :



En bleu l'immunoglobuline prédite et en vert Spike Wuhan 8DV1.

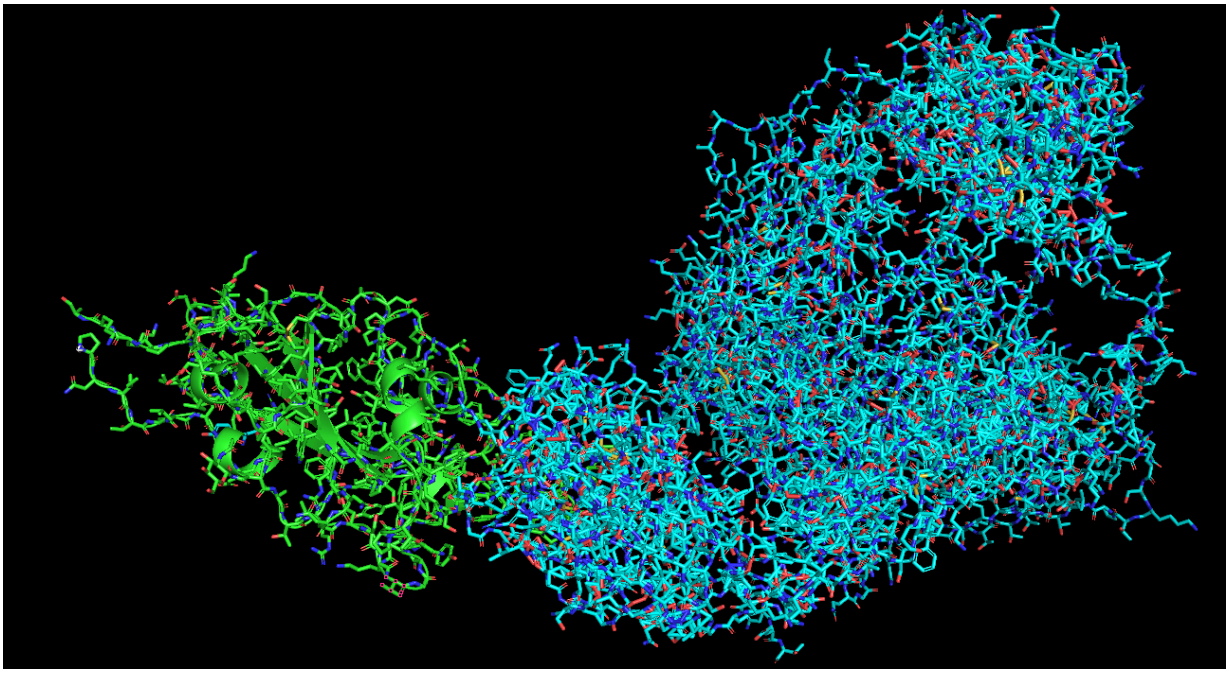
Une fois le calcul fait avec HDOCK, environ 10 minutes, on obtient un nouveau fichier au format PDB et un fichier .out contenant l'ensemble des énergies de liaison.

```
Grid spacing:      1.200
Angle step:     15.000
Initial rotation:      0.00000    0.00000    0.00000
rec_67a29d9709717.pdb      254.579    257.279    278.132
lig_67a29d9709717.pdb        1.688      0.000     10.000
    5.95560    1.89911    2.06522    241.121    286.796    327.205    -379.92     492.66       1.00
    3.47562    1.26285    5.19029    240.187    286.907    327.239    -379.34     491.51       1.00
    5.96391    2.06772    2.09493    247.971    284.133    331.198    -377.50     497.26       1.00
    6.02729    1.92862    2.11752    244.707    288.834    327.701    -377.20     496.09       1.00
    6.01073    2.13426    2.40545    255.878    286.691    332.419    -374.35     503.82       1.00
    3.46505    1.21153    5.16597    241.107    285.236    327.718    -353.07     491.31       1.00
```

On voit donc ici que pour nos meilleurs docking on a des énergies de liaison vers -370 kcal/mol, ce qui est un résultat encourageant car selon la littérature c'est similaire à ce que l'on devrait obtenir.

Test

En testant sur la première immunoglobuline de notre fichier CSV et en faisant toute la pipeline on obtient le meilleur docking ci-dessous :



Récapitulatif

Le projet a donc était globalement fini car il y a une pipeline qui va permettre de tester si une immunoglobuline présente un intérêt quant à une protéine cible, pathogène, en partant de séquences d'acides aminés pour finir sur un docking moléculaire avec calcul d'énergies de liaison.

Continuation et ouverture

J'ai bien aimé faire ce projet qui demandait beaucoup plus de réflexions dans le côté biologie, choix et adaptation d'outils existants, que dans une partie code pur. Mais le sujet et l'application de ce projet est extrêmement motivant et intéressant, car on ose imaginer l'impact que notre projet pourrait avoir pour tous les patients du *Streptococcus agalactiae.*

Même si les résultats semblent satisfaisants, j'émet cependant une limite notable à ceux-ci : ce n'est pas parce que le docking est stable qu'il y a un effet neutralisant du pathogène. Je ne sais donc pas s'il est réalisable d'obtenir grâce à cette pipeline dans son état actuel, une liste d'immunoglobulines pouvant neutraliser le pathogène.

Le projet a pour but d'être continuer dans un avenir proche pour tester sur un cluster de GPU afin d'expérimenter la pipeline sur tout le fichier CSV de test afin de valider celle-ci s'il y a correspondance, dans une certaine mesure, avec les résultats obtenus en laboratoire. Si la pipeline est valide, alors on pourra imaginer une adaptation de celle-ci pour *Streptococcus agalactiae.*

ANNEXES

Bibliographie

https://www.uniprot.org/
https://github.com/google-deepmind/alphafold
https://deepmind.google/discover/blog/alphaproteo-generates-novel-proteins-for-biology-and-health-research/
https://onlinelibrary.wiley.com/doi/10.1002/pro.3978

Guide d'installation d'AlphaFold

Pour installer AlphaFold 2

Tout d'abord il faut avoir un environnement Linux natif (pas de VM ni WSL car bug possible de driver ou perte de performance).

Personnellement j'ai utilisé Linux Mint 22 (basé sur Ubuntu 24.04)

Il faut ensuite se rendre sur le github fait par Google Deepmind : https://github.com/google-deepmind/alphafold

On peut voir qu'il faut installer Docker sur Linux. Je ne conseille pas de prendre la version Desktop car elle va utiliser de précieuses ressources.

- Installation de Docker :

Dans le terminal : sudo apt-get update; sudo apt install -y docker.io; sudo systemctl start docker; sudo systemctl enable docker; sudo usermod -aG docker $USER

Cela va permettre d'installer Docker et de permettre de le lancer sans les privilèges root.

- Installation des dépendances NVIDIA :

AlphaFold ne fonctionne qu'avec des cartes graphiques Nvidia et avec un bon PC.
Ma configuration de PC : Intel Core i5-12400f, 32Gb RAM DDR4, 1To SSD, NVIDIA RTX 4060Ti 16Gb.

Tout d'abord sur Linux je conseille de désactiver le Secure Boot (dans le BIOS), car il peut rentrer en conflit avec les drivers de la carte graphique.

Maintenant je conseille d'installer les drivers recommandés pour la carte graphique (nvidia-driver-550 pour ma part). Une fois l'installation faite il faut installer le NVIDIA Container Toolkit : https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html

J'ai personnellement fait via Apt en suivant les commandes indiquées sur le site :
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \

&& curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | \

sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' | \

sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list

sudo apt-get update; sudo apt-get install -y nvidia-container-toolkit

- Clone le repo :

Une fois dans le dossier souhaité :

git clone https://github.com/deepmind/alphafold.git

- Installer aria2c :

sudo apt install aria2

- Installation des bases de données nécessaires à AlphaFold :

- Installer cuda :

sudo apt install nvidia-cuda-toolkit

Personnellement, sur ma carte graphique c'est Cuda

Si erreur (mon cas) lorsqu'on essaie de lancer docker run --rm --gpus all nvidia/cuda:11.0-base nvidia-smi

```
antoine@antoine-MS-7D98:~$ docker run --rm --gpus all nvidia/cuda:11.0-base-ubuntu24.04 nvidia-smi
Unable to find image 'nvidia/cuda:11.0-base-ubuntu24.04' locally
docker: Error response from daemon: manifest for nvidia/cuda:11.0-base-ubuntu24.04 not found: manifest unknown: manifest unknown.
See 'docker run --help'.
```

Essayer les fix suivants :

- https://github.com/google-deepmind/alphafold/issues/805 : `docker run --rm --gpus all nvidia/cuda:11.0.3-base-ubuntu20.04 nvidia-smi`

- Sinon, se rendre sur : https://developer.nvidia.com/cuda-11.0-download-archive?target_os=Linux&target_arch=x86_64&target_distro=Ubuntu&target_version=2004&target_type=runfilelocal et saisir les commandes sur le prompt dans l'arborescence : Linux → x86_64 → Ubuntu → 20.04 → runfile (local)

Download Installer for Linux Ubuntu 20.04 x86_64

The base installer is available for download below.

Base Installer

Installation Instructions:

```
$  wget http://developer.download.nvidia.com/compute/
cuda/11.0.2/local_installers/
cuda_11.0.2_450.51.05_linux.run
$  sudo sh cuda_11.0.2_450.51.05_linux.run
```

Une fois qu'un résultat similaire à la photo ci-dessous s'affiche, docker aura accès au GPU.

```
antoine@antoine-MS-7D98:~$ docker run --rm --gpus all nvidia/cuda:11.0.3-base-ubuntu20.04 nvidia-smi
Wed Jan 22 16:15:17 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.120                Driver Version: 550.120        CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 4060 Ti     Off |   00000000:01:00.0  On |                  N/A |
|  0%   40C    P8             11W /  165W |     726MiB /  16380MiB |      8%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
+-----------------------------------------------------------------------------------------+
```

Normalement à ce stade l'installation de toutes les dépendances est faite.

Je conseille alors de faire toutes les étapes du Guide Python avant de lire la suite.

Maintenant que Python est installé et que notre environnement virtuel est configuré, nous allons pouvoir construire l'image Docker d'AlphaFold.

- Se rendre dans le répertoire alphafold du projet.

- Exécuter la commande suivante : docker build -f docker/Dockerfile -t alphafold .

Si l'erreur suivante apparaît :

```
The following packages will be UPDATED:

  conda                           24.11.1-py311h06a4308_0 --> 24.11.3-py311h06a4308_0


Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done
Channels:
 - conda-forge
 - defaults
 - nvidia
Platform: linux-64
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... failed

InvalidSpec: The package "nvidia/linux-64::cuda-compiler==12.6.3=0" is not available for the specified platform

The command '/bin/bash -o pipefail -c conda install --quiet --yes conda==24.11.1 pip python=3.11     && conda install --quiet --yes
--channel nvidia cuda=${CUDA_VERSION}     && conda install --quiet --yes --channel conda-forge openmm=8.0.0 pdbfixer     && conda cl
ean --all --force-pkgs-dirs --yes' returned a non-zero code: 1
```

- Vérifier que conda est bien installé.

- Si c'est le cas alors faire l'étape suivante :

- conda search -c nvidia cuda-compiler

Guide d'installation Python

Créer son environnement virtuel python sur linux :

- Installer python via le cmd
- sudo apt install python3.12-dev
- sudo apt install python3.12-venv
- python3 -m venv my_env
- source ~/my_env/bin/activate
- pip install numpy
- pip install pandas
- pip install scipy
- pip install nglview
- pip install pytest
- pip install graphein
- pip install requests
- pip install swig
- pip install openbabel-wheel (permet d'avoir openbabel précompilé)
- pip install subprocess.run
- pip install -r PycharmProjects/AntibodiesIntelligence/alphafold/docker/requirements.txt

Pour l'IDE j'ai utilisé Pycharm Professionel. Il faut donc configuré ajouter notre environnement virtuel : File → Settings → Projects → Python Interpreter → Add

Interpreter → Add local Interpreter → Select existing → Path_du_venv/bin/python3.12

Il faut également installer Conda. Pour cela, je vais installer Miniconda.

- wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

- bash Miniconda3-latest-Linux-x86_64.sh

Pour installer Pymol :

- sudo apt-get install pymol

Si conflit (car version 3.12 de python par exemple):

```
antoine@antoine-MS-7D98:~$ pymol
Traceback (most recent call last):
  File "<frozen runpy>", line 189, in _run_module_as_main
  File "<frozen runpy>", line 112, in _get_module_details
  File "/usr/lib/python3/dist-packages/pymol/__init__.py", line 81, in <module>
    from imp import find_module
ModuleNotFoundError: No module named 'imp'
```

Il faut faire :

- sudo nano /usr/lib/python3/dist-packages/pymol/__init__.py

Puis modifier la ligne “from imp import find_module” par “from importlib.util import find_spec as find_module”

On va utiliser la librairie PyMOL-PUB, sauf que lorsqu’on essaie de l’installer on a l’erreur suivante :

```
(my_env) antoine@antoine-MS-7D98:~$ pip install pymol-pub==1.2
Collecting pymol-pub==1.2
  Using cached PyMOL_PUB-1.2-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: numpy in ./my_env/lib/python3.12/site-packages (f
rom pymol-pub==1.2) (1.26.4)
INFO: pip is looking at multiple versions of pymol-pub to determine which versio
n is compatible with other requirements. This could take a while.
ERROR: Could not find a version that satisfies the requirement pymol (from pymol
-pub) (from versions: none)
ERROR: No matching distribution found for pymol
```

Cela vient du fait que le module python de PyMol n’est pas installé. On va donc utiliser PyMol à partir du chemin d’installation actuel:

- which pymol

Cela renvoie “/usr/bin/pymol”

Installer PyMol via le GitHub et faire l’import Python :

- git clone https://github.com/schrodinger/pymol-open-source.git

- sudo apt-get install build-essential python3-dev libglew-dev libglm-dev freeglut3-dev libpng-dev libfreetype6-dev libxml2-dev libmsgpack-dev libnetcdf-dev qt5-qmake qtbase5-dev

- git clone https://github.com/rcsb/mmtf-cpp.git

- cd mmtf-cpp

- mkdir build

- cd build

- cmake ..

- make

- sudo make install

- cd

- cd pymol-open-source

- source ~/my_env/bin/activate

- python3 setup.py build --glut=yes

- python3 setup.py install

Normalement PyMol devrait être installé dans notre environnement virtuel Python, et la partie interface graphique également.

Pour installer AutoDockVina:

- Télécharger le fichier .tgz : https://vina.scripps.edu/downloads/

- cd Downloads

- tar -xzf autodock_vina_1_1_2_linux_x86.tgz

- export PATH=$PATH:$(pwd)/autodock_vina_1_2_5_linux_x86_64/bin

- sudo mv autodock_vina_1_1_2_linux_x86/bin/vina /usr/local/bin/

Vérifier avec :

- which vina

- vina --version