FACULTAD DE CIENCIAS AGRARIAS

UNIVERSIDAD NACIONAL DE ROSARIO

MultimerMapper: Una Herramienta para el Análisis y Visualización de Datos Generados con AlphaFold2-multimer

Bioq. Elvio Rodriguez Araya

TRABAJO FINAL PARA OPTAR AL TITULO DE ESPECIALISTA EN BIOINFORMÁTICA

DIRECTOR: Dr. Rodolfo Rasia

CODIRECTOR: Dr. Esteban Serra

2024

Elvio Rodriguez Araya

Bioquímico – Universidad Nacional de Rosario

Este Trabajo Final es presentado como parte de los requisitos para optar al grado académico de Especialista en Bioinformática, de la Universidad Nacional de Rosario y no ha sido previamente presentada para la obtención de otro título en esta u otra Universidad. El mismo contiene los resultados obtenidos en investigaciones llevadas a cabo en la Facultad de Ciencias Agrarias - UNR, durante el período comprendido entre marzo y julio del 2024, bajo la dirección de Rodolfo Rasia y la codirección de Esteban Serra.

Bioq. Elvio Rodriguez Araya

Dr. Rodolfo Rasia

Dr. Esteban Serra

Defendida: ..... de 2024.

Publicaciones y Presentaciones a Congresos

- XIII CAB2C - 13th Argentinian Conference in Bioinformatics and Computational Biology. XIII SoIBio - 13th International Conference of the Iberoamerican Society of Bioinformatics. III Annual Meeting of the Ibero-American Artificial Intelligence Network for Big BioData. 1 al 3 de noviembre de 2023. Rosario, Prov. de Santa Fe, Argentina. **Sequential Assembly of Large Bromodomain Containing Complexes in Trypanosomatids**. Elvio Rodriguez Araya; Esteban Serra.

1. Abreviaturas y Símbolos

IA: Inteligencia Artificial

AF2: AlphaFold2

AF2m: AlphaFold2-multimer

MDa: Mega Daltons

HPC: Computación de Alto Rendimiento

PPI: Interacciones Proteína-Proteína

PyPI: Índice de Paquetes De Python

MSA: Alineamiento Múltiple

2-mero: Dímero

N-mero: Oligómero

IDE: Entorno de Desarrollo Integrado

pLDDT: Test de Diferencia Local Predicho

pTM: Puntaje de Modelado con Plantilla Predicho

ipTM: pTM de Interfase

PAE: Error Alineado Predicho

ROC: Receiver Operating Characteristic

FPR: Tasa de Falsos Positivos

MM: MultimerMapper

RMSD: Root Mean Square Deviation

Å: Angstroms

CoIP: Coinmunoprecipitación

MultimerMapper: Una Herramienta para el Análisis y Visualización de Datos Generados con AlphaFold2-multimer

Resumen: El presente trabajo se centra en la lógica detrás del desarrollo de MultimerMapper, una herramienta innovadora diseñada para el análisis y visualización de datos obtenidos con AlphaFold2-multimer. MultimerMapper facilita la exploración detallada de interacciones proteína-proteína en complejos, utilizando un enfoque que captura la variabilidad estructural de estas biomoléculas. Mediante el uso de técnicas avanzadas de bioinformática y visualización, se ha logrado integrar la información proveniente de múltiples modelos estructurales para crear representaciones visuales interactivas en 2D y 3D, situando las superficies de interacción y los contactos residuo-residuo como eje central. MultimerMapper se presenta como una herramienta prometedora en el campo de la bioinformática estructural, ofreciendo nuevas perspectivas para futuras investigaciones sobre la dinámica de interacciones dentro de complejos proteicos de orden superior.

Palabras Clave: AlphaFold2, Interacciones Proteína-Proteína, Complejos.

MultimerMapper: A Tool for the Analysis and Visualization of AlphaFold2-multimer Interaction Landscapes

Summary: This work focuses on the logic behind the development of MultimerMapper, an innovative tool designed for the analysis and visualization of data obtained with AlphaFold2-multimer. MultimerMapper facilitates detailed exploration of protein-protein interactions in complexes, using an approach that captures the structural variability of these biomolecules. Using advanced bioinformatics and visualization techniques, information from multiple structural models has been integrated to create interactive visual representations in 2D and 3D, with special focus in interaction surfaces and residue-residue contacts. MultimerMapper is presented as a promising tool in the field of structural bioinformatics, offering new perspectives for future research on the dynamics of interactions within higher-order protein complexes.

Keywords: AlphaFold2, Protein-Protein Interactions, Complexes.

Índice

2. Introducción

En los últimos años, el desarrollo de algoritmos de inteligencia artificial (IA) con la capacidad de resolver problemas de alta complejidad ha sido exponencial. En el contexto de la bioinformática, uno de los hitos más importante fue el desarrollo del modelo de *deep learning* **AlphaFold2** (AF2) (Jumper, 2021), capaz de predecir con alta precisión la estructura de las proteínas en su forma monomérica (**Fig. 1A**). Una versión posterior, **AF2-multimer** (AF2m) (Evans, 2021) fue reentrenada para predecir la estructura de múltiples proteínas en simultaneo, es decir, polímeros de proteínas (**Fig. 1B**), dando paso a la predicción de interacciones proteína-proteína a partir de las superficies de contacto entre las subunidades predichas (Bryant, 2022).

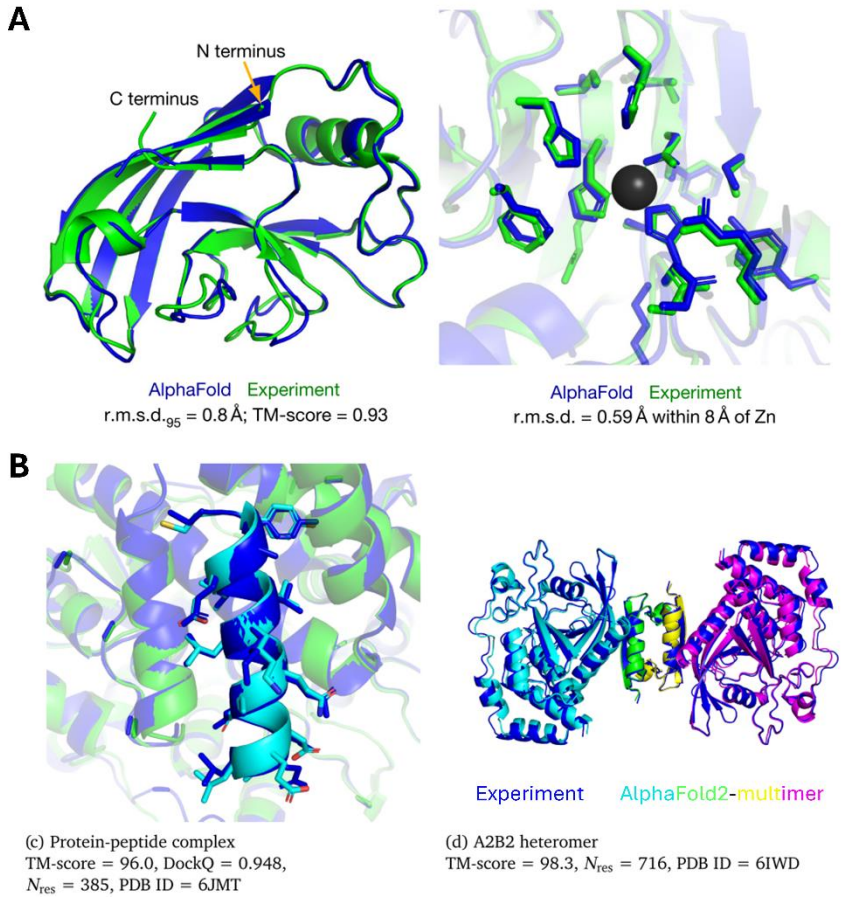

Fig. 1: Ejemplos de predicciones con AlphaFold2 (AF2) y AlphaFold2-multimer (AF2m). (**A**) Predicciones de monómeros (AF2) con precisión atómica. (**B**) Predicciones de heterómeros (AF2m) con precisión atómica. Adaptado de Evans et al (2021) y Jumper et al. (2021).

A su vez, la implementación de estos modelos en plataformas de cómputo en la nube como el Colaboratorio de Google (Colab), ha permitido democratizar su uso en la

comunidad científica, donde cualquiera puede ejecutar estos modelos utilizando como *input* tan solo las secuencias de las proteínas que se deseen estudiar. La versión de AF2 de ejecución en la nube más difundida es la de ColabFold (Mirdita, 2022). Dadas las limitaciones del cómputo en la nube proporcionada por Colab, se desarrolló una versión de ejecución local a través de la línea de comando de Linux (https://github.com/YoshitakaMo/localcolabfold) que conserva la versatilidad de ColabFold, ampliando aún más su alcance y utilización.

A la hora de analizar complejos proteicos de gran tamaño, sus limitaciones en el uso de memoria al manejar grandes cantidades de residuos de aminoácido hicieron evidente la necesidad de enfoques complementarios. Por ejemplo, metodologías no basadas en IA, como el algoritmo de ensamblaje combinatorio implementado en **CombFold** (Shor, 2024), facilitan la predicción de complejos de orden superior (escala de los MDa). Estas metodologías se basan en fragmentar el problema en combinaciones superpuestas de predicciones, para luego integrar la información parcial y generar un modelo global del complejo (**Fig. 2**).

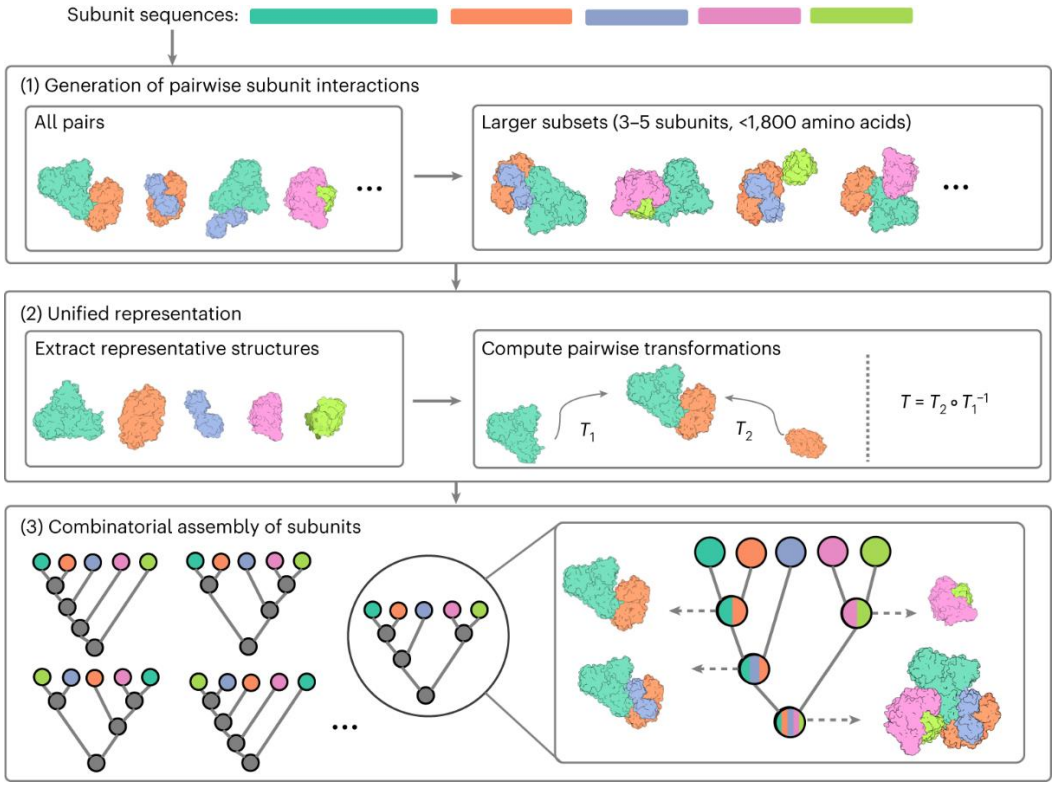

Fig. 2: Etapas implementadas durante el ensamblado combinatorio implementado por CombFold. (**1**) Se predice con AF2m todas las interacciones diméricas posibles entre todas las subunidades del complejo proteico, seguido por predicciones solapadas de trí,tetra,penta-meros. (**2**)

El algoritmo extrae estructuras representativas de cada subunidad y computa todas las posibles transformaciones de a pares. (**3**) El ensamblado combinatorio se realiza de forma jerárquica en forma de árboles de ensamblaje donde se exploran todos los caminos. Extraído de Shor (2024).

Si bien la información generada para complejos pequeños es fácilmente manejable, el estudio in silico de grandes complejos proteicos sigue siendo un desafío, ya que no existen marcos de análisis de datos de predicción de estructuras proteicas derivadas de IA. Esto se vuelve evidente cuando las predicciones se aplican a un nivel de computación de alto rendimiento (HPC). Además, uno de los grandes desafíos continúa siendo la correcta inferencia de los coeficientes estequiométricos de los componentes del complejo que se busca ensamblar (Shor, 2024). Otro desafío no explorado consiste en que, entre las decenas de proteínas que interactúan entre sí, muchas pueden hacerlo a través de interacciones transitorias o dinámicas que ocupan las mismas superficies de contacto, lo que hace imposible su existencia en simultaneo. Un claro ejemplo de este comportamiento son las proteínas de unión a dominios MRG (MRGBP) de eucariotas (**Fig. 3**), donde múltiples proteínas compiten por la misma superficie de interacción sobre un mismo dominio MRG (Xie, 2015).

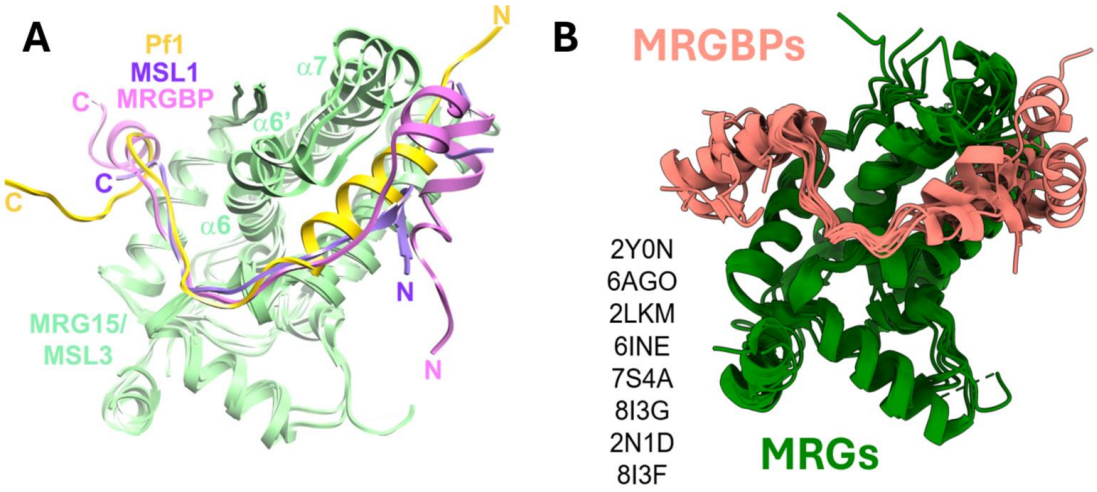

Fig. 3: Multiespecificidad de los dominios MRG. (**A**) Resumen gráfico del artículo donde se describe por primera vez la promiscuidad de los dominios MRG utilizando cristalografía de rayos X (Xie et al., 2015). Tres proteínas distintas interactúan sobre la misma superficie del dominio MRG de la proteína MRG15 de humanos. (**B**) Nueve años más tarde continúan detectándose nuevos interactores de dominios MRG, todos interactuando sobre la misma superficie. Se muestran los identificadores de PDB de las estructuras superpuestas utilizadas.

Según mis observaciones, los modelos generados con AF2-multimer manifiestan o no estas interacciones dinámicas dependiendo del contexto. Es decir, algunas

interacciones aparecen o desaparecen dependiendo de las proteínas introducidas en el modelo y de sus coeficientes estequiométricos (cuantas unidades de cada proteína). Los algoritmos de ensamblado combinatorio actuales no tienen esta información en cuenta, sino que simplemente se limitan a ensamblar el complejo que el usuario indica. Así, esta valiosa información es ignorada por desconocimiento o no es aprovechada para inferir comportamientos moleculares que podrían ser relevantes a nivel biológico. En mi opinión, estas dificultades radican en la ausencia de herramientas bioinformáticas de integración y análisis *a posteriori* de datos generados por AF2m. Por lo tanto, en este trabajo final se describe el desarrollo de **MultimerMapper**, una herramienta computacional que facilita la integración, la visualización y el análisis de conjuntos de predicciones generadas por AF2m. En particular, esta herramienta es de gran utilidad para analizar el comportamiento de interacciones proteína-proteína (PPI) entre las subunidades que conforman los complejos proteicos de gran tamaño. Este programa utiliza la teoría de grafos para extraer, analizar y visualizar la información contenida en las estructuras y archivos generados por AF2m, entregando representaciones en 2D y 3D de los contactos existentes dentro de las redes de interacción, junto a su dinamismo contextual.

3. Objetivos

Objetivo General

Desarrollar una librería de Python que permita analizar y generar representaciones biológicamente interpretables, principalmente en forma de grafos, a partir de datos generados con AlphaFold2-multimer para conjuntos de proteínas contenidas en complejos proteicos.

Objetivos Específicos

- Desarrollar funciones/clases para analizar las coordenadas atómicas contenidas en los conjuntos de archivos PDB generados con AF2-multimer para grandes complejos proteicos.
- Desarrollar funciones/clases para analizar las métricas contenidas en los conjuntos de archivos JSON asociados a archivos PDB generados con AF2-multimer para grandes complejos proteicos.

- Desarrollar funciones/clases que conviertan la información generada en información visual de 2 y 3 dimensiones, la cual será exportada en archivos HTML que se ejecuten en cualquier navegador (incluyendo dispositivos móviles).
- Documentar todas las funciones y clases dentro de los módulos generados.
- Integrar todo en una librería de Python llamada MultimerMapper para su posterior distribución a través del índice de Paquetes de Python (PyPI).
- Comprobar el correcto funcionamiento de la librería utilizando set de datos de prueba generados con AlphaFold2-multimer para conjuntos de proteínas contenidas en complejos proteicos.

4. Materiales y Métodos

Implementación de AlphaFold2-multimer y alineamientos múltiples

Todas las estructuras fueron predichas utilizando la versión 3 (v3) de AF2m implementada en LocalColabFold versión 1.5.2 (https://github.com/YoshitakaMo/localcolabfold). Las ejecuciones de los datos provenientes de complejos proteicos se realizaron utilizando instancias de Amazon Web Services (AWS) g4dn.12xlarge, g5.48xlarge, p3.2xlarge, p3.16xlarge o p3dn.24xlarge, dependiendo del tamaño del conjunto de datos y del número total de residuos de las predicciones. En este caso LocalColabFold se ejecutó con las siguientes opciones: `--num-models 5 --num-recycle 20 --rank ipTM --recycle-early-stop-tolerance 0.5 --num-relax 1 --use-gpu-relax`. Para los datos de entrenamiento/testeo, ver abajo.

Los alineamientos múltiples (MSA) de entrada se generaron llamando al servidor de MMseqs2 (Mirdita, 2022) utilizando las secuencias completas de las proteínas a modelar. Además, se enriquecieron con secuencias del grupo taxonómico discoba (Wheeler, 2021) de la siguiente manera. Primero, para permitir el apareamiento de secuencias de la misma especie, se anotó la base de datos original de Discoba con identificadores taxonómicos numéricos (TaxID). Para hacerlo, se agrupó los encabezados de las secuencias FASTA por su prefijo común más largo y se añadió un TaxID único a cada secuencia de cada grupo con el formato `|TaxID=Discoba001`

y se guardó con el nombre `discoba_TaxID.fasta`. Se la transformó en una base de datos de MMseqs2 usando el comando `mmseqs createdb discoba_TaxID.fasta discoba` y luego se indexó con `mmseqs createindex discoba tmp`. La búsqueda de secuencias homólogas para cada secuencia de consulta individual `query.fasta` se realizó utilizando los siguientes comandos:

```
mmseqs createdb <query.fasta> queryDB
mmseqs search queryDB discoba resultsDB tmp
mmseqs align queryDB discoba resultsDB alignDB -a
mmseqs convertalis queryDB discoba alignDB queryDB.tab --format-output
target,qlen,qstart,qend,tstart,tend,tseq,cigar,taln
```

La tabla de salida de MMseqs2 se reformateó a A3M y se añadió la secuencia de consulta al inicio del alineamiento resultante.

Para el emparejamiento de secuencias, se siguió el procedimiento descrito por Zhou et al (Zhou, 2018). Brevemente, las secuencias en cada alineamiento individual se agruparon por su TaxID de Discoba y se clasificaron de mayor a menor similitud de secuencia con la secuencia de consulta utilizando el algoritmo de Needleman-Wunsch (Needleman & Wunsch, 1970). Para cada grupo, las secuencias del mismo rango se emparejaron entre sí. Aquellas sin par en el mismo rango se descartaron de la parte emparejada del MSA, así como aquellas que tenían al menos una secuencia con menos del 50% de cobertura. La parte no emparejada del MSA se construyó añadiendo tantos huecos al final o al principio de las secuencias como la longitud de la(s) otra(s) secuencia(s). Finalmente, se añadió una línea de cardinalidad al principio del archivo A3M con el formato $\#L_1,L_2,\ldots,L_n\ 1,1,\ldots,1$ donde L_i son las longitudes de cada secuencia. El MSA resultante se concatenó sobre el final del archivo MSA producido con el servidor ColabFold MMseqs2.

Set de datos de entrenamiento/testeo

Con el objetivo de encontrar los mejores valores de corte de las métricas generadas por AF2m que permitan discriminar mejor los pares de proteínas que interactúan entre si (positivos) de los que no (negativos), se generó un conjunto de datos de entrenamiento/testeo. El subconjunto positivo se construyó mediante el curado manual de la literatura, buscando pares de proteínas de tripanosomátidos con

evidencia *in vivo* o *in vitro* de interacciones directas. La evidencia experimental provino principalmente de experimentos de doble híbrido en levadura con proteínas de *T. brucei*, *T. cruzi* y *Leishmania*, resultando en una lista de 175 pares de proteínas interactuantes. Para generar el subconjunto negativo, se crearon listas de proteínas de tripanosomátidos pertenecientes a distintos complejos eucariotas conocidos. Luego, se combinaron seleccionando aleatoriamente una proteína de un complejo y otra de un complejo diferente, asegurando que tuvieran una baja probabilidad de interactuar entre sí. Por ejemplo, se seleccionó una proteína aleatoria del complejo del poro nuclear y una proteína aleatoria del complejo de la zona de unión del flagelo. Este proceso se repitió hasta obtener 200 pares de proteínas. El código utilizado para generar el *dataset* negativo puede encontrarse en el siguiente repositorio de GitHub: https://github.com/elviorodriguez/DiscobaMultimer/tree/main/acetylation_related_complexes/benchmark_dataset_generation.

Los pares fueron modelados AF2m v3 y las siguientes opciones de ColabFold: `--num-models 5 --num-recycle 3 --rank ipTM --stop-at-score 75 --recycle-early-stop-tolerance 2.5 --save-all`. Se estableció un límite de 2000 residuos, lo que resultó en 169 complejos predichos para el conjunto de datos positivo y 161 para el conjunto de datos negativo. Las ejecuciones se realizaron en dos servidores de acceso remoto con placas Nvidia RTX 3090 (Servidor 1: 24 GB VRAM, Ryzen 3700X, 64 GB RAM; Servidor 2: 24 GB VRAM, Inter Core i9-12900F, 16 GB RAM).

Set de datos de complejos

Para verificar el correcto funcionamiento del programa y sus funciones en un contexto de aplicación real, se recuperaron diversas listas de proteínas contenidas en complejos proteicos nucleares relacionados con la acetilación de histonas de *T. brucei* a partir de la literatura y se procesaron como se describe a continuación. Para cada complejo proteico, se generaron los modelos diméricos de todas las combinaciones posibles (incluyendo homodímeros) con AF2m de las proteínas que los componen (2-meros). Para aquellos pares identificados como interactores (ver más abajo), se expandieron las predicciones de AF2m para cubrir trímeros, tetrámeros, pentámeros, etc. (N-meros), a modo de capturar información de dinamismo contextual.

Entorno de desarrollo de MultimerMapper

Los módulos y paquetes del programa fueron desarrollados en los entornos de desarrollo integrado (IDE) Spyder y Visual Studio codo, usando Anaconda 3 como manejador de paquetes, Git como software de control de versiones y GitHub como repositorio para el almacenamiento en línea del sistema de archivos necesarios para ejecutar y desarrollar el programa. La versión de Python utilizada fue 3.11. Como interfaz entre el sistema operativo y Python, se utilizó el paquete “os”. La lectura de los archivos JSON se realizó utilizando el paquete “json”. Para el análisis de expresiones regulares se utilizaron los paquetes “string”, “re” y “difflib”. Se utilizó el paquete “BioPython” para la manipulación de archivos PDB, FASTA y las coordenadas atómicas. Las operaciones matemáticas de objetos vectoriales se realizaron utilizando el paquete “NumPy” y la manipulación de datos tabulares se realizó utilizando el paquete “Pandas”. Los gráficos bidimensionales no interactivos fueron generados utilizando el paquete “Matplotlib” y los gráficos interactivos se generaron utilizando el paquete “Plotly”. Se utilizó el paquete “py3Dmol” para la representación estructuras proteicas de forma interactiva.

Ensamblado combinatorio

Para realizar el ensamblado combinatorio de los complejos proteicos se utilizó CombFold (https://github.com/dina-lab3D/CombFold), siguiendo la misma metodología descrita por Shor (2024), con algunas modificaciones. Brevemente, a partir de las estequiometrías más probables inferidas utilizando las salidas de MultimerMapper, se generaron los archivos JSON que definen las subunidades para ensamblar combinatoriamente utilizando la salida AF2m para las listas de proteínas de complejos proteicos (2-meros y N-meros) como base de datos de modelos para el ensamblado. A diferencia de lo descrito por Shor (2024), se utilizaron las secuencias completas de las proteínas para realizar los modelos de AF2m de las proteínas y luego se las fragmentó en subunidades, en lugar de trabajar directamente con los fragmentos.

5. Resultados y Discusión

Con el objetivo de facilitar la interpretación del lector, se detallará en primer lugar la estructura de los datos de entrada de AF2 (*input*), sus datos de salida (*output*) y el

algoritmo del estado-del-arte utilizado para ensamblar complejos proteicos de gran tamaño. Posteriormente, se describirá lo que aquí se propone como “hipótesis de las interacciones dinámicas” y una solución algorítmica que posibilita su “mapeo” dentro de un marco computacional que las considere. Esta solución implica clasificar pares de proteínas como interactoras/no interactoras, por lo que se mostrará la optimización de parámetros utilizando un set de prueba para lograr esta clasificación. Luego, se describe el proceso de desarrollo de MultimerMapper y la lógica implementada. Por último, se utilizan como ejemplo algunos complejos nucleares de tripanosomátidos para describir como MultimerMapper puede facilitar la extracción de información contenida en conjuntos de predicciones de AF2m y asistir en la inferencia de estequiometrías para realizar un correcto ensamblado combinatorio.

El *input* y el *output* de AlphaFold2 y AlphaFold2-multimer

En la **Fig. 4** puede verse un esquema del flujo de predicción de estructuras de complejos con AF2m. Como *input* se requieren las secuencias de las proteínas que se desean modelar en conjunto, en un formato similar al FASTA. Este archivo se procesa con un módulo de alineamiento que recupera secuencias homólogas a las secuencias de consulta para generar un alineamiento múltiple (MSA). El MSA consta con una sección desapareada y otra apareada. La sección apareada contiene los alineamientos de cada proteína aislada. La sección apareada une las secuencias de las distintas proteínas homólogas, pero solo de aquellas que pertenecen a la misma especie. Se vio que este tipo de alineamiento es el que mejor captura la información evolutiva para lograr una buena calidad de plegado individual de cada proteína (intra-cadena), a la vez que se captura la información de contactos inter-cadena (Bryant, 2022).

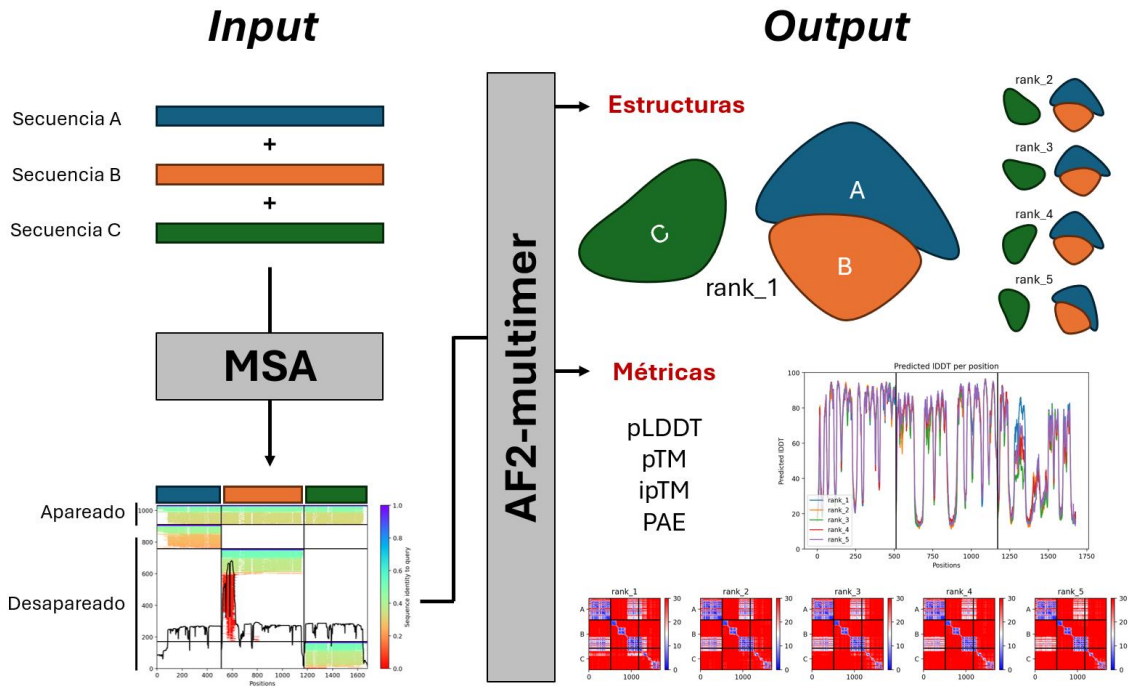

Fig. 4: *Input* y *output* de AF2m.

El *output* cuenta con las predicciones estructurales *per se* (archivos PDB), y una serie de métricas asociadas a cada estructura dentro de archivos JSON. Estas métricas son estimaciones estadísticas de métricas normalmente utilizadas en el análisis de estructuras de proteínas. El **pLDDT** (test de diferencia local predicho) es un valor informado por residuo (valores entre 0 y 100) que otorga una idea de la confianza que existe en la posición del carbono α de cada residuo a nivel local. Esto puede interpretarse como el nivel de confianza de la estructura secundaria asignada a cada residuo. Mientras más alto este valor, más confianza. El pLDDT promedio de cada proteína da una estimación de que tan bien fue predicha su estructura. El **pTM** (puntaje de modelado con plantilla predicho) se informa globalmente y da una idea de la confianza existente en el plegado entre partes distantes de las proteínas. En general, valores de pTM altos (cercanos a 1) indican estructuras compactas y valores bajos (cercanos a 0) indican estructuras desplegadas o desordenadas. El **ipTM** (pTM de interfase) representa la contribución al pTM únicamente de las interfases entre proteínas, es decir, de las zonas donde las proteínas interactúan. Mientras mayor sea la confianza en la predicción de una interfase, más alto será el valor. Otra métrica es el PAE (error alineado predicho), la más importante a la hora de discriminar pares interactores de no interactores. En la **Fig. 5** se muestra un ejemplo de la matriz PAE para un sistema de 3 proteínas que interactúan entre sí. La matriz es cuadrada y

aproximadamente simétrica. Ambos ejes representan las posiciones (Nº de residuo) de cada proteína y el valor dentro para cada par de residuos representa el error predicho por AF2 entre las posiciones relativas de sus átomos en la estructura final. Es decir, si dos residuos tienen bajo error alineado predicho, existe alta confianza en sus posiciones relativas dentro de la estructura. Cuando se analizan pares de residuos de la misma cadena, dan indicios de la presencia de dominios globulares. Para el ejemplo, se pueden ver 3 dominios globulares en la proteína B (recuadros y flechas verdes). Al analizar pares de residuos inter-cadena, estos valores nos dan indicios de la presencia de interacciones entre las cadenas. Por ejemplo, entre los pares de residuos del dominio globular 1 de la proteína B y los residuos de la proteína A (rectángulos de puntos naranjas) existe un alto error alineado predicho, por lo que se puede interpretar que el dominio globular 1 no interactúa con A. Por otro lado, los residuos del dominio globular 3 de B tiene bajo error alineado predicho con A (rectángulos de puntos violetas), por lo que se puede interpretar que este dominio se encuentra en contacto con la proteína A.

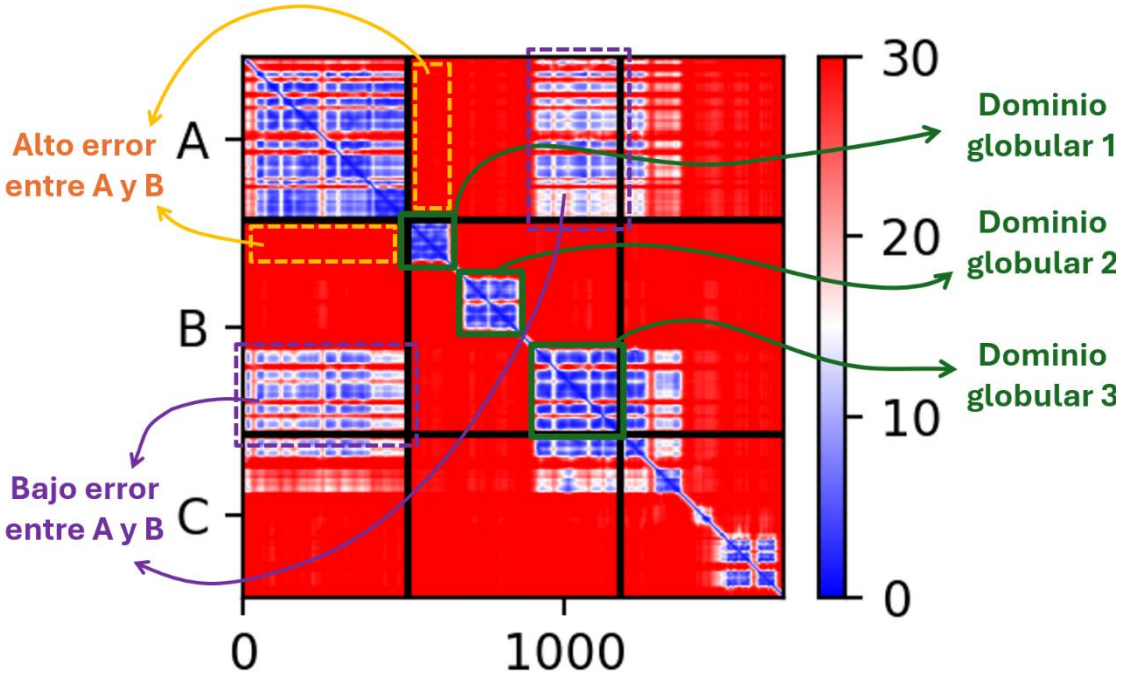

Fig. 5: Matriz PAE para un sistema de 3 proteínas que interactúan a través de distintos dominios. Los colores de la matriz representan el error alineado predicho (PAE) en Angstroms.

Por último, es necesario destacar que AF2m puede generar el número de modelos que uno desee. Típicamente, se generan 5 modelos, que a su vez son rankeados

utilizando alguna de las métricas descritas (rank_1, rank_2, etc.). Cuando se trabaja con PPIs, se utiliza el ipTM como métrica de escalafonado.

El algoritmo de ensamblado combinatorio

A la hora de analizar grandes complejos proteicos, muchas veces es imposible ejecutar AF2m utilizando todas las proteínas que integran el complejo en simultaneo. Esto se debe principalmente a una limitación computacional, ya que el tiempo de cómputo es proporcional al cuadrado del número de residuos del modelo (Shor, 2024). Como se mencionó en la introducción, la solución se encontró en el ensamblado combinatorio (**Fig. 6**).

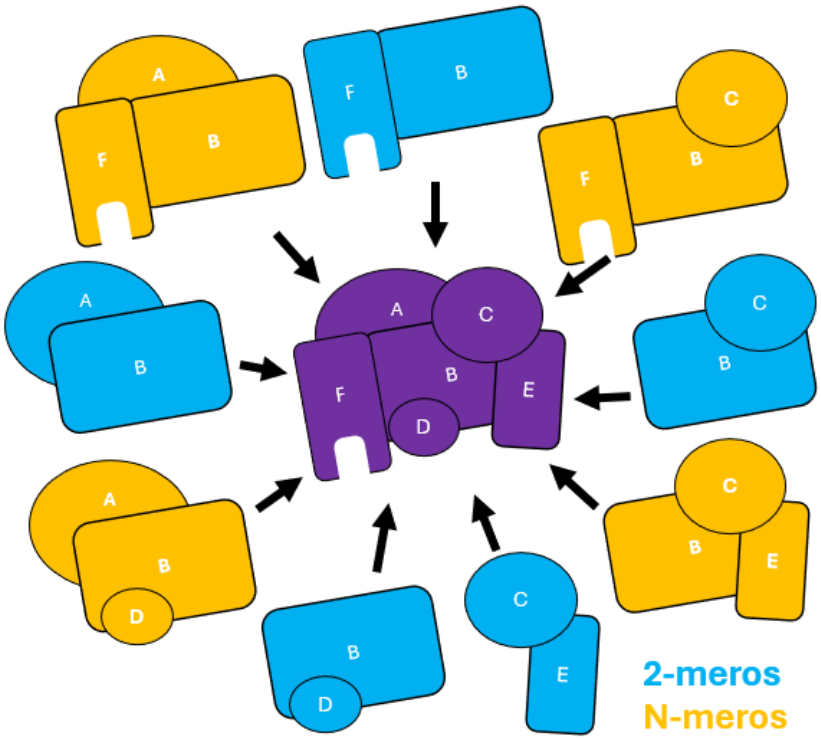

Fig. 6: Ensamblado combinatorio. Los dímeros (2-meros) y los tri-, tetra-, pentámeros (N-meros) se computan primero con AF2m, para luego ensamblar el complejo de orden superior utilizando el algoritmo de ensamblado combinatorio.

En primer lugar, la metodología requiere realizar todas las combinaciones de predicciones diméricas posibles (2-meros, léase dí-meros). En teoría, es posible utilizar únicamente los 2-meros para realizar un correcto ensamblado. Sin embargo, los autores de CombFold, observaron empíricamente que la calidad del ensamblado mejora significativamente si se realiza un paso extra de predicciones que involucre combinaciones solapadas de tri-, tetra- y pentámeros (N-meros, léase oligó-meros).

Esto se debe a que distintas regiones de las proteínas pueden interactuar con distintas proteínas en simultaneo, por lo que se pueden ver afectadas las conformaciones locales y globales al establecerse interacciones en simultaneo. El algoritmo de ensamblado combinatorio se aplica sobre estos modelos (2-meros y N-meros), indicándole a CombFold los coeficientes estequiométricos de cada subunidad. El algoritmo integra la información parcial de los subcomplejos para ensamblar el complejo de orden superior, alcanzando calidades iguales o superiores a la obtenida utilizando supercomputadoras que ejecuten AF2m con todas las subunidades en simultaneo.

Los problemas del ensamblado combinatorio, la hipótesis de las interacciones dinámicas y una posible solución

Si bien CombFold resuelve las limitaciones computacionales, es una metodología de punto final que fue desarrollada usando ejemplos ideales donde las subregiones de las proteínas involucradas en la formación del complejo eran conocidas, así como también sus coeficientes estequiométricos. Es decir, es necesario conocer mucho sobre el sistema para poder obtener modelos biológicamente relevantes. Además, CombFold no proporciona representaciones que indique lo que sucede con las superficies de interacción, sino que se limita a encontrar el mejor camino de ensamblado utilizando una estructura representativa por subregión de las proteínas (subunidades con mayor pLDDT promedio), que son estáticas, sin tener en cuenta si los coeficientes estequiométricos suministrados son los “correctos”. Por lo tanto, si el sistema es poco conocido, sería necesario recurrir a aproximaciones experimentales para obtener información sobre los coeficientes estequiométricos de los componentes, justamente la premisa que desea evitarse en el campo de la bioinformática estructural debido a su laboriosidad técnica.

Para ahondar en la problemática y describir una posible solución, supongamos que estamos estudiando un sistema simple de 3 proteínas (A, B y C), para el cual solo sabemos que formarían un complejo y contamos con las predicciones de sus estructuras monoméricas (**Fig. 7**).

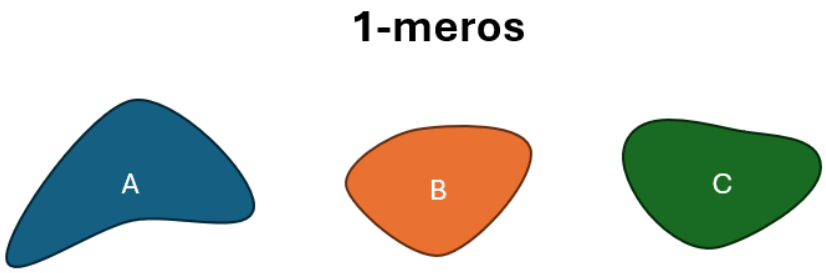

Fig. 7: Monómeros de un sistema de tres proteínas.

Si queremos ensamblar el complejo, será necesario computar primero los 2-meros. Comenzamos con los hetero-2-meros y observamos manualmente que las métricas y las estructuras de AF2m indican que A interacciona con B y C, mientras que B y C no interaccionan entre si (**Fig. 8**). Además, notamos que la superficie de interacción de A en AB es la misma que en AC.

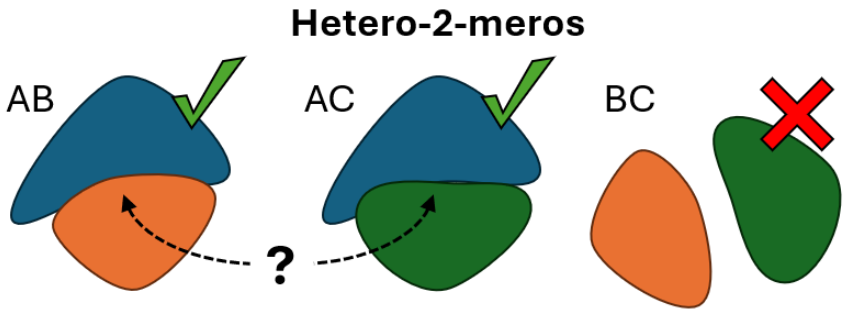

Fig. 8: Todas las combinaciones posibles de heterodímeros del sistema de tres proteínas. La superficie de interacción de A con B y C es la misma ¿Qué está sucediendo?

Podríamos preguntarnos ¿Qué está sucediendo? Seguramente AF2m no anda tan bien y una de las interacciones es un falso positivo ¿No? Ya que no existe manera que ambas interacciones se den en simultaneo. No obstante, por lo que se discutió en la introducción (**Fig. 3**), sabemos de la existencia de interacciones moleculares promiscuas y quizá estemos ante una situación similar. Entonces, podríamos computar el hetero-3-mero para ver que interacción prevalece, si es que alguna (**Fig. 9**).

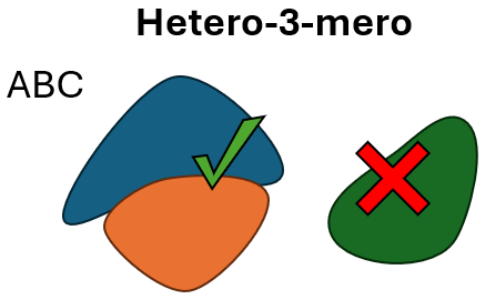

Fig. 9: Heterotrímero del sistema de tres proteínas. Prevalece la interacción AB.

En este caso, vemos que prevalece únicamente la interacción AB, mientras que la interacción AC "desaparece". Si bien estamos discutiendo sobre un modelo hipotético, no es inusual encontrar esta situación. Según mi experiencia, AF2m tiende a decidirse por una de las dos interacciones. Por lo tanto, desde el punto de vista de los modelos de AF2m, podríamos decir que la interacción AC es **DINÁMICA** (presente en el 2-mero y presente en el 3-mero) y la interacción AB es **ESTÁTICA** (presente en el 2-mero, pero ausente en el 3-mero). Sin embargo, para acceder a esta información fue necesario explorar los modelos, sus superficies de contacto, y analizar las métricas (principalmente las matrices PAE), y así deducir que es lo que está sucediendo ¿No sería conveniente contar con una herramienta que detecte automáticamente este tipo de comportamiento en los conjuntos de modelos? Para ello, primero sería necesario contar con un clasificador que discrimine pares de proteínas interactoras de no-interactoras, tanto en los modelos diméricos, como en los N-méricos. También sería necesario contar con un marco computacional que permita su sistematización. Para construir este marco, podemos recurrir a la teoría de grafos, representando las proteínas como **vértices** (*nodes*) y las superficies de interacción como **aristas** (*edges*). Para el ejemplo, por un lado, podemos representar las PPIs del conjunto de 2-meros, obteniendo las aristas A-C y B-C. Por el otro, la representación en forma de grafo del 3-mero contendrá solo la arista A-B. Al comparar ambos grafos podemos decir que la interacción A-C desaparece en los N-meros (es dinámica) y la interacción A-B prevalece (es estática) (**Fig. 10**).

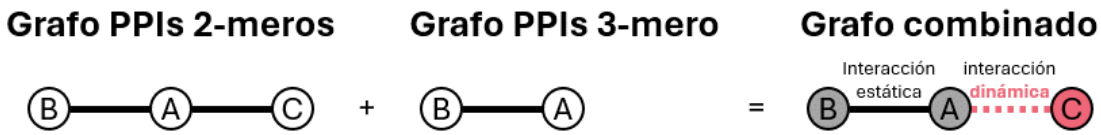

Fig. 10: Representación en forma de grafos de los modelos que contamos hasta el momento para el sistema de 3 proteínas. Al comparar los grafos, podemos identificar que existe una interacción dinámica.

En teoría, las situaciones en que pueden manifestarse interacciones dinámicas no se limitan a situaciones donde existen superficies compartidas, sino que también puede explicarse por cambios conformacionales inducidos por otras proteínas que activan (dinámica positiva) o inhiben (dinámica negativa) interacciones con una tercera proteína. En este sentido, por "**dinámica**" y "**estática**" nos referimos al comportamiento de la PPI al comparar ambos grafos. Así, podemos construir un sistema de 5 reglas que permiten clasificar las interacciones y las proteínas involucradas según su comportamiento (**Fig. 11**).

Algo para remarcar es que con este enfoque no necesitamos observar manualmente los modelos para deducir esta información. Más bien, AF2m decide qué interacción prevalece con su conocimiento de la fisicoquímica de proteínas y el análisis de grafos la extraería. Mientras tanto, la exploración manual de los modelos se limitaría únicamente a la explicación del mecanismo que provoca la interacción dinámica, no a su detección. A pesar de ello, cuando múltiples modelos de N-meros contienen la misma proteína bajo muchas condiciones diferentes, las posibilidades dejan de ser binarias (interacción dinámica/estática), ya que la misma interacción puede aparecer en un contexto y desaparecer en otro. Por lo tanto, el lector debe anticipar que será necesario extender las reglas para considerar el dinamismo en forma de espectro. De cualquier manera, estas reglas pueden aplicarse a conjuntos de modelos de proteínas mucho más grandes, como ya veremos.

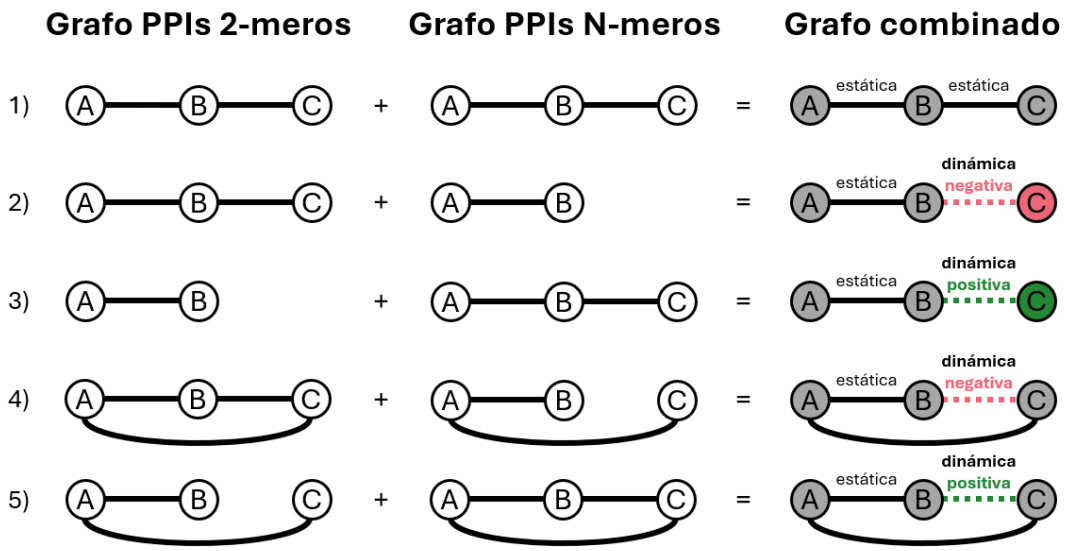

Fig. 11: Lógica de clasificación de PPIs en un sistema de tres proteínas. (**1**) Ambas interacciones están presentes en ambos grafos. Las proteínas e interacciones son estáticas. (**2**) Ambas interacciones están presentes en el grafo de 2-meros, pero solo la arista A-B está presente en el grafo de N-meros. Por lo tanto, C es una proteína dinámica y su interacción con B es dinámica. (**3**) La arista A-B está presente en ambos grafos y la arista B-C aparece solo en el grafo de N-meros. Este es un caso de activación de interacción inducida por la presencia de A. (**4**-**5**) Mismo caso que 2 y 3, respectivamente, pero involucrando una tercera interacción A-C. En estos casos, C está presente en ambos grafos combinados porque participa en una PPI con A, manteniendo este nodo dentro del grafo final. Esto hace que la proteína C sea estática, pero no su interacción con B.

El análisis no debería detenerse en este punto, ya que restan aspectos que deben introducirse en nuestro marco computacional, indispensables para la correcta inferencia de los coeficientes estequiométricos de cada componente. En particular, introducir el comportamiento de las **homo**-interacciones (**Fig. 12**).

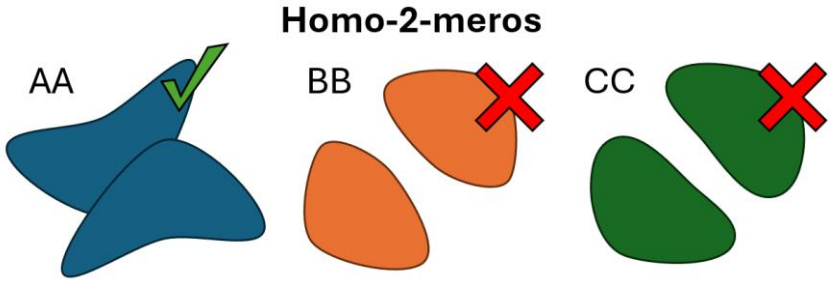

Fig. 12: Homodímeros del sistema de 3 proteínas. Solo A homodimeriza.

Al analizar los homo-2-meros, observamos que solo A genera modelos que homodimerizan. El lector atento puede que ya esté suponiendo lo siguiente: la

homodimerización de A resulta en una estructura simétrica que expone dos potenciales superficies de interacción de A con B y C (vea la **Fig. 8**). La clave está en confirmar la suposición al expandir los homo-2-meros de A con distintas combinaciones de 3-meros (**Fig. 13**).

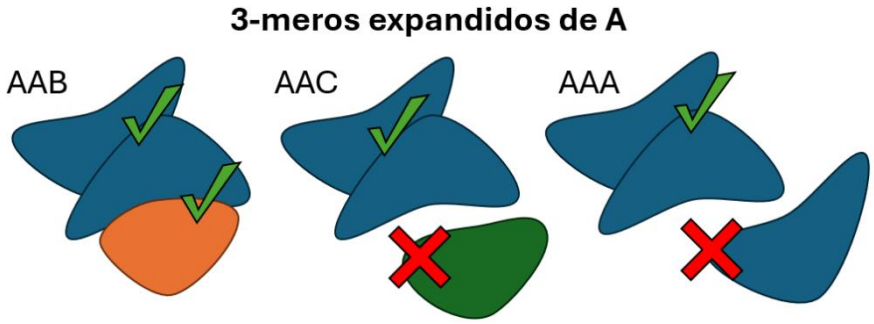

Fig. 13: Trímeros expandidos del homodímero AA para el sistema de 3 proteínas. A no forma homotrímeros, pero si heterotrímeros con B, aunque no con C.

Intrigantemente, los trímeros indican que el homodímero AA puede interactuar con B, pero no con C y no formaría homooligómeros de orden superior al homodímero. Nuevamente ¿Qué está sucediendo? Antes de sacar conclusiones, integremos la homodimerización al marco computacional. Esto lo podemos lograr utilizando aristas que conecten nodos consigo mismos (**Fig. 14**).

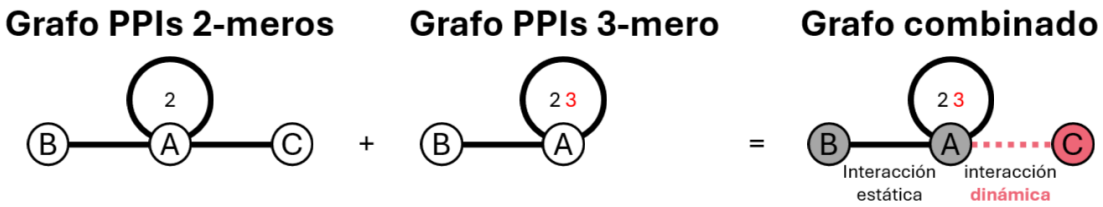

Fig. 14: Representación con grafos del sistema de tres proteínas que contempla la homooligomerización. La arista circular indica que A puede homooligomerizar. El 2 de color negro indica que la homodimerización es positiva y el 3 en rojo indica que la homotrimerización es negativa.

Esta representación ahora contiene información sobre los estados de homooligomerización de las proteínas en los 2-meros y en los N-meros. Al integrarlo en el grafo combinado, sabemos que A homodimeriza, pero no homotrimeriza y que existe una interacción dinámica negativa entre A y C. Subamos un nivel más y computemos algunos 4-meros relevantes (**Fig. 15**).

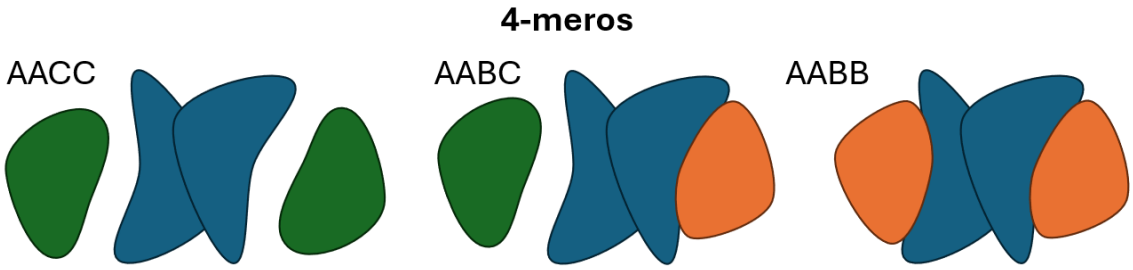

Fig. 15: Tetrámeros AACC, AABC y AABB del sistema de tres proteínas.

Al parecer encontramos una estequiometría con 2 subunidades de A y 2 subunidades de B, ya que C consistentemente no interactúa con las formas homodiméricas de A. Retomando con la pregunta (¿Qué está sucediendo?), una hipótesis funcional podría involucrar a C como regulador de la interacción AB, donde la unión de C solo a la forma monomérica de A no sería coincidencia, ya que teóricamente respondería al principio de Le Chatelier (desplazamiento del equilibrio). Es decir, en un contexto celular (ya sea un compartimento particular, una región de la cromatina, etc.) donde C se encuentre en cantidades abundantes, C tendría la capacidad de unirse a la forma monomérica de A y retrasar la interacción prematura de B con el monómero de A, suponiendo que la forma funcional del complejo requiere que A homodimerice. No obstante, sea cual sea la explicación mecanística, sabemos que la interacción A-C no se manifiesta en oligómeros de orden superior al heterodímero. Por lo tanto, la proteína C no sería parte del complejo de orden superior (su coeficiente estequiométrico sería de cero). Esto resulta contraintuitivo desde el punto de vista de, por ejemplo, los experimentos de inmunoprecipitación que buscan recuperar proteínas interactoras utilizando *tags*. Imaginemos que conducimos uno de estos experimentos marcando la proteína A con un *tag* para coinmunoprecipitar todos sus interactores y recuperamos sistemáticamente a la proteína C. La lógica indicaría que esta proteína debería formar parte del complejo. Incluso si conducimos experimentos de interacción directa entre A y C, estos resultaran positivos. Lo que sucede es que los comportamientos dinámicos de las interacciones permiten la coexistencia de dos entidades distintas que dependen del contexto.

Aquí podremos suponer que alcanzamos nuestra deseada estequiometría, pero ¿Es suficiente la cantidad de modelos con la que contamos y sus combinaciones? ¿Cuándo debemos detenernos? La adición de modelos debería continuar al menos

hasta que las nuevas combinaciones no introduzcan nuevas interacciones ni cambien la clasificación dinámica de otras interacciones. En otras palabras, hasta alcanzar la convergencia (**Fig. 16**).

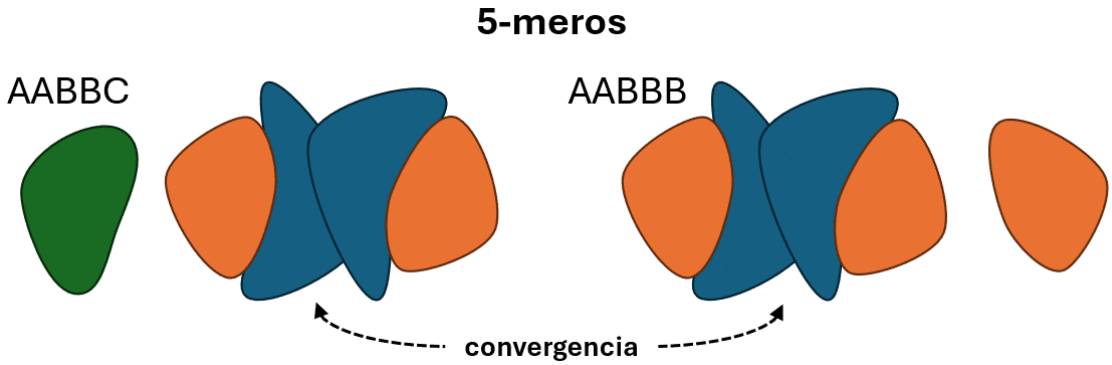

Fig. 16: Pentámeros relevantes para el sistema de tres proteínas. En este punto se alcanza la convergencia, ya que volvemos a obtener estructuras observadas en N-meros de orden inferior.

El lector debe tener en cuenta que esta es una representación hipotética de un sistema relativamente sencillo de tres componentes. La lógica aplicada en el ejemplo puede extenderse a sistemas más grandes para obtener información relevante sobre las interacciones dinámicas, los estados de homooligomerización posibles y sus combinaciones, permitiendo en última instancia inferir estequiometrías. A la vez, no hay que perder de vista que la idea de complejos proteicos estáticos muchas veces es inconsistente con lo observado y deben pensarse como elementos dinámicos que se ensamblan y desensamblan a través de caminos que involucran múltiples estequiometrías. Entonces, los pasos para generar un set de datos que permita obtener las interacciones dinámicas e inferir estequiometrías son:

1. Predecir todas las combinaciones de hetero- y homo-2-meros.
2. Verificar quienes son los interactores. Proteínas que no interactúen deben retirarse.
3. Realizar todas las combinaciones hetero-3-meros. Si no hay homodímeros, omitir las combinaciones que posean proteínas repetidas.
4. Si no hay homo-2-meros. Detenerse.
5. Si hay alguna proteína que homotrimeriza, expandir con 4-meros que incluyan los homotrímeros, e incluir los homo-4-meros.
6. Repetir los pasos 2 a 5, subiendo un nivel más cada vez que se obtenga un homopolímero de orden superior.
7. Detenerse al alcanzar la convergencia.

Esto resulta en un número mínimo de predicciones diméricas ($P_{2-meros}$) que depende del Nº de proteínas del sistema (N) según:

$$P_{2-meros} = \frac{N\ (N+1)}{2} \qquad \textbf{Ecuación 1}$$

Mientras, el número de predicciones de los N-meros ($P_{N-meros}$) va a variar desde cero, cuando no haya proteínas que interactúen en el sistema, hasta infinito, cuando no se alcance la convergencia:

$$0 \leq P_{N-meros} \leq \infty \qquad \textbf{Ecuación 2}$$

Esta última situación, teóricamente, podría ocurrir cuando existan proteínas cuyos patrones de interacción generen modelos homo/heterooligoméricos sin convergencia, como son las proteínas involucradas en la formación de polímeros filamentosos lineales (heterodímero α/β-tubulina, por ejemplo). Omitiendo estos casos, según mi experiencia, el número de modelos de N-meros necesarios generalmente es de un orden similar o inferior al Nº de 2-meros. En los casos de geometrías no convergentes, sería prudente detenerse al identificarlos.

Al "mapear" estas interacciones dentro de un grafo que aplique la lógica aquí descrita, podremos representar las PPIs y su comportamiento. Nuevamente, como notará, si se desea automatizar el proceso es necesario contar con un clasificador que discrimine pares de proteínas **interactoras** (+) de **no-interactoras** (-) e incorporarlo en forma de módulo dentro del programa.

Optimización de *cutoffs*

Para entrenar el módulo clasificador, fue necesario generar un *dataset* de prueba/entrenamiento. Para ello, se llevó a cabo un curado manual de la literatura, recuperando pares de proteínas de tripanosomátidos para las cuales se haya obtenido evidencia experimental de que establecen interacciones directas (**Fig. 17**). El resultado fueron 175 pares de proteínas interactoras.

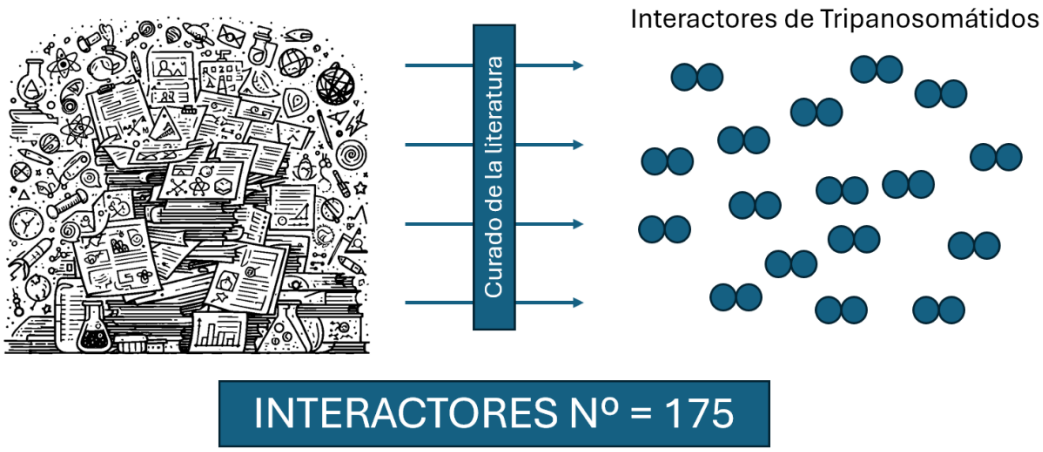

Fig. 17: Generación del *dataset* positivo. Los pares de proteínas interactoras se obtuvieron mediante el curado manual de la bibliografía de tripanosomátidos.

El *dataset* negativo se generó mediante la siguiente aproximación. Se generaron listas de proteínas pertenecientes a diferentes complejos eucariotas conservados también en tripanosomátidos y se las etiquetó según el nombre del complejo y el compartimento subcelular. Se definieron pares de complejos con funciones no relacionadas y compartimentos subcelulares separados, de forma tal que sus componentes tengan muy baja probabilidad de poseer pares de proteínas interactoras. Se tomó al azar un par de complejos y por cada complejo se eligió una proteína al azar para generar un par no-interactor. El proceso se repitió hasta alcanzar 200 pares, sin duplicados (**Fig. 18**).

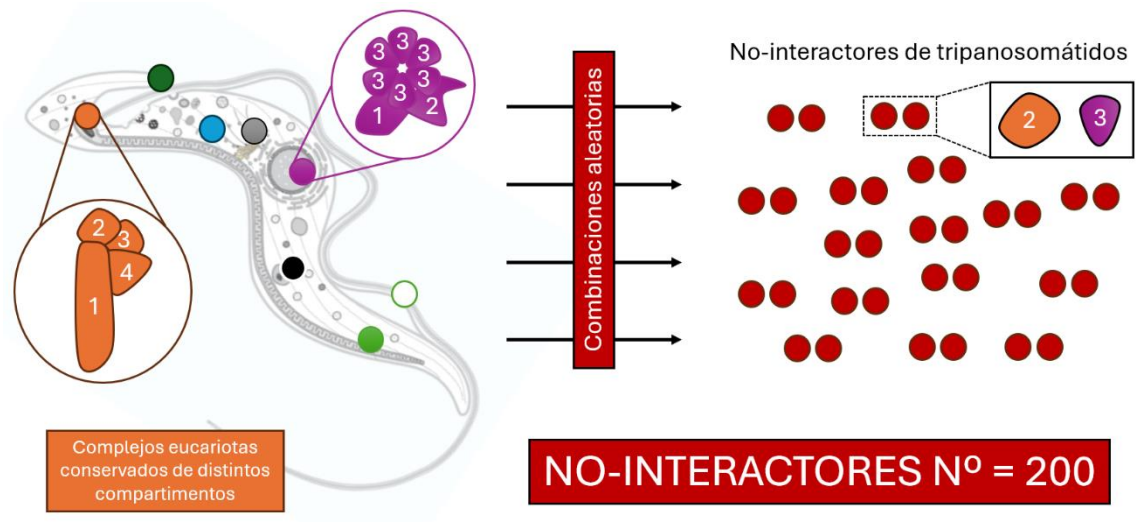

Fig. 18: Generación del *dataset* negativo. Los pares de proteínas no-interactoras se generaron mediante la combinación aleatoria de proteínas de diferentes complejos conservados con baja probabilidad de contener pares interactores.

Ambos *dataset* se combinaron y se predijeron los dímeros con AF2m, utilizando los valores de ipTM como método de *ranking* (**Fig. 19**). Solo se computaron los pares cuyo número combinado de aminoácidos no superaba los 2000, lo que resultó en un número final de 169 positivos y 161 negativos. A cada predicción se le extrajo las métricas asociadas (pLDDT, pTM, ipTM, PAE) y se computaron dos métricas extra relacionadas directamente con la probabilidad de que el modelo represente una interacción. Una fue el pDockQ, ampliamente utilizada y validada para la predicción de interacciones (Bryant, 2022). pDockQ involucra el análisis de los valores de pLDDT por residuo sobre las interfaces proteína-proteína y es una estimación de la métrica DockQ, utilizada para medir la calidad de modelos estructurales de interacción generados mediante metodologías de *docking* (Basu, 2016). La otra métrica es un parámetro que disminuye la dimensionalidad de la matriz PAE, concentrándose en la region de interacción. Este es el valor mínimo del error predicho alineado para la interacción (miPAE, por sus siglas en inglés, *minimum interaction PAE*). Como su nombre lo indica, se obtiene hallando el valor mínimo de error en la matriz PAE para los pares de residuos de distintas cadenas (el valor mínimo de los conjuntos de residuos interproteína de la **Fig. 5**).

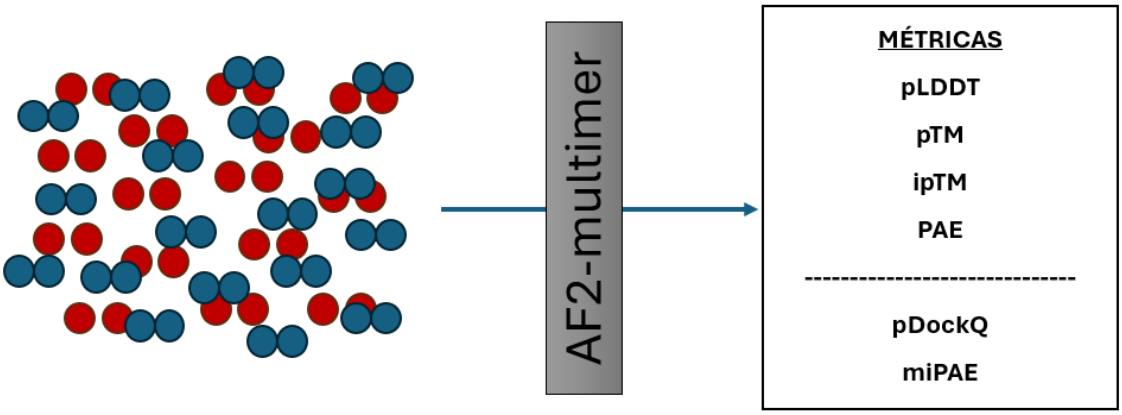

Fig. 19: Predicción de estructuras del *dataset* de prueba/entrenamiento. El complejo de cada par fue predicho con AF2m y sus métricas fueron extraídas.

El objetivo es encontrar los valores de corte de estas métricas que maximicen la sensibilidad a la vez que se minimicen las inespecificidades. Se decidió trabajar únicamente con las métricas involucradas en la detección de interfaces (ipTM, pDockQ y miPAE). Se separaron las métricas en 2 pares, ipTM + miPAE y pDockQ + miPAE, para encontrar la mejor combinación de valores de corte conjuntos. La idea es utilizar la combinación ipTM + miPAE en modelos provenientes de 2-meros y la combinación pDockQ + miPAE en dímeros provenientes de N-meros. Esto se debe a que el ipTM es una métrica asociada a la totalidad del modelo, por lo que su contribución a la evaluación de interacciones específicas es menos precisa en modelos donde se involucren múltiples proteínas. En cambio, pDockQ se centra en la calidad de la interacción entre dos proteínas, las cuales se pueden aislar de los N-meros para computar la métrica solo entre ellas, haciéndola más adecuada para dímeros en el contexto de N-meros.

Para determinar los valores óptimos de corte, se realizó un análisis de curvas ROC (*Receiver Operating Characteristic*) variando los valores de corte conjuntamente, a la vez que se varió el número necesario de modelos dentro de cada predicción que debían superar esta métrica. Los perfiles de sensibilidad máxima para distintos valores de tasas de falsos positivos (FPR) y para distintos números mínimos de modelos se muestran en la **Fig. 20**.

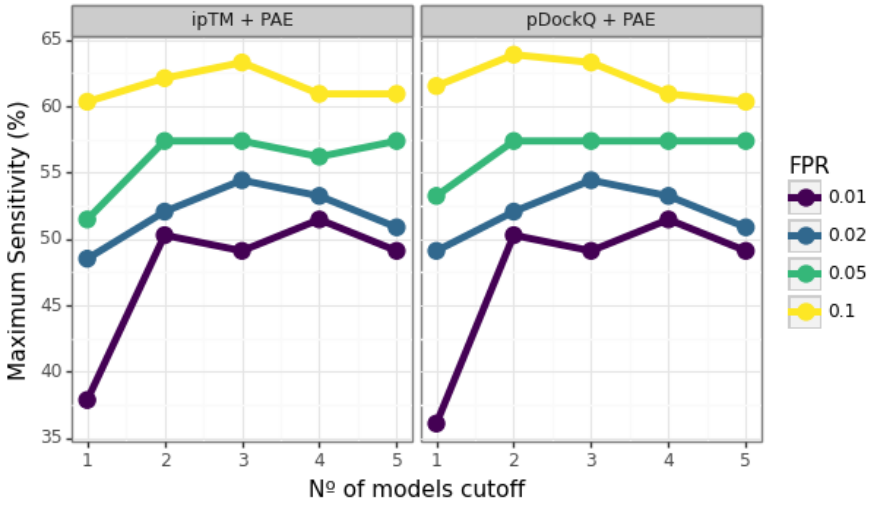

Fig. 20: Perfiles de sensibilidad máxima a distintas tasas de falsos positivos para distintos números valores de cortes de modelos. (Izquierda) Sensibilidades máximas de la combinación ipTM + miPAE. (Derecha) Sensibilidades máximas de la combinación pDockQ + miPAE.

Se puede observar que, en general, la sensibilidad tiende a maximizarse usando los *cutoffs* optimizados con 2 o 3 modelos como corte. Se decidió trabajar a una FPR de 0.05, utilizando 3 modelos como valores de corte. Esto resultó en una sensibilidad máxima del 57.4% con ambos modelos (**Fig. 21**).

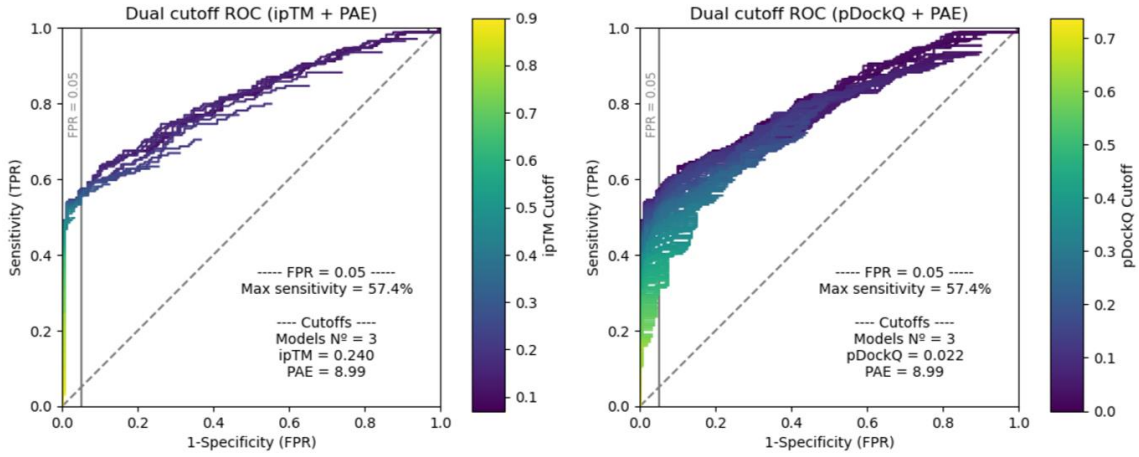

Fig. 21: Curvas ROC asociadas a cada combinación de métricas para un valor mínimo de modelos de 3. La línea vertical gris señala la tasa de falsos positivos de 0.05.

Al comparar estos resultados con trabajos de revistas de alto prestigio conducidos sobre set de datos *gold standard* para la predicción de PPIs, la metodología aplicada al set de datos generado de este trabajo alcanza resultados similares. Por ejemplo, la publicación en Nature Comunications de Bryant (Bryant, 2022) reporta una

sensibilidad máxima del 51% a una FPR de 0.01, comparada con la sensibilidad máxima obtenida de 51.5% a la FPR de 0.01 de esta metodología (**Fig. 20**). Por lo tanto, podemos estar seguros de que contamos con un clasificador competitivo para discriminar PPIs en tripanosomátidos.

Desarrollo de MultimerMapper

La idea detrás del desarrollo de MultimerMapper no es solo obtener las estequiometrías de complejos incógnita, sino extraer la mayor cantidad posible de información sobre conjuntos de predicciones de oligómeros proteicos generados con AF2m para últimamente combinarla con otros *pipelines*. El centro de atención está sobre las superficies de interacción y el comportamiento de las PPI, pero también sería deseable que detecte dominios, aminoácidos involucrados en las PPI, cambios conformacionales inducidos por interacciones, entre otras características. Además, dado que cada usuario generará su set de datos según sus capacidades y necesidades, el programa tiene que ser capaz de detectar errores en las predicciones, o incluso situaciones donde la información contenida en los conjuntos de predicciones no es suficiente para estimar el comportamiento dinámico de las interacciones, sugiriendo así combinaciones extra. Todo esto, a su vez tiene que ser computacionalmente interpretable (la computadora tiene que entenderlo) y suministrarse de forma "digerible" para un humano, es decir, a través de visualizaciones que permitan entender intuitivamente el comportamiento subyacente de las proteínas en los modelos.

Puesto que el proceso de desarrollo de un software es un proceso iterativo de prueba y error que involucra múltiples ciclos de diseño, en esta sección solo se describirá la lógica implementada en cada uno de los módulos, mientras que el código fuente de MultimerMapper (MM a partir de ahora) puede encontrarse en mi repositorio de GitHub (https://github.com/elviorodriguez/MultimerMapper). En la próxima sección se muestra como ejecutar el programa y su *output* con ejemplos prácticos, usando *datasets* de complejos nucleares incógnita de tripanosomátidos.

En la **Fig. 22** se puede ver un esquema del flujo de información de la primera sección de MM y las funcionalidades clave de cada módulo. El programa recibe como *input* mínimo un archivo FASTA con las secuencias de aminoácidos de las proteínas que están presentes en las predicciones y un directorio conteniendo todas las predicciones

de 2-meros (ambos homo- y heterodiméricos). Cada secuencia del archivo FASTA debe contener como encabezado (*header)* los identificadores únicos de las proteínas y un nombre o símbolo único para facilitar su seguimiento. Opcionalmente, se puede suministrar los N-meros, habilitando la detección de interacciones dinámicas. Esto está pensado para un flujo de trabajo iterativo donde el usuario inicialmente genera el *dataset* de 2-meros (vea la **Ecuación 1**) y ejecute MM con estos datos para detectar los pares de proteínas de proteínas interactores. Posteriormente, el usuario generará las combinaciones de N-meros pertinentes (vea la **Ecuación 2**) y volverá a ejecutar MM adicionando esta información. MM será capaz de identificar interacciones dinámicas y dará aviso al usuario si falta adicionar información (faltan computar N-meros). El proceso debe repetirse hasta alcanzar la convergencia. El archivo FASTA, los directorios con 2-meros y con N-meros son analizados por un primero módulo que verifica el correcto formato del sistema de archivos. Si el formato es correcto, los archivos se cargan en la memoria RAM en forma de variables utilizando los paquetes “json” y “BioPython”, y se transfieren al módulo de extracción de métricas de MM.

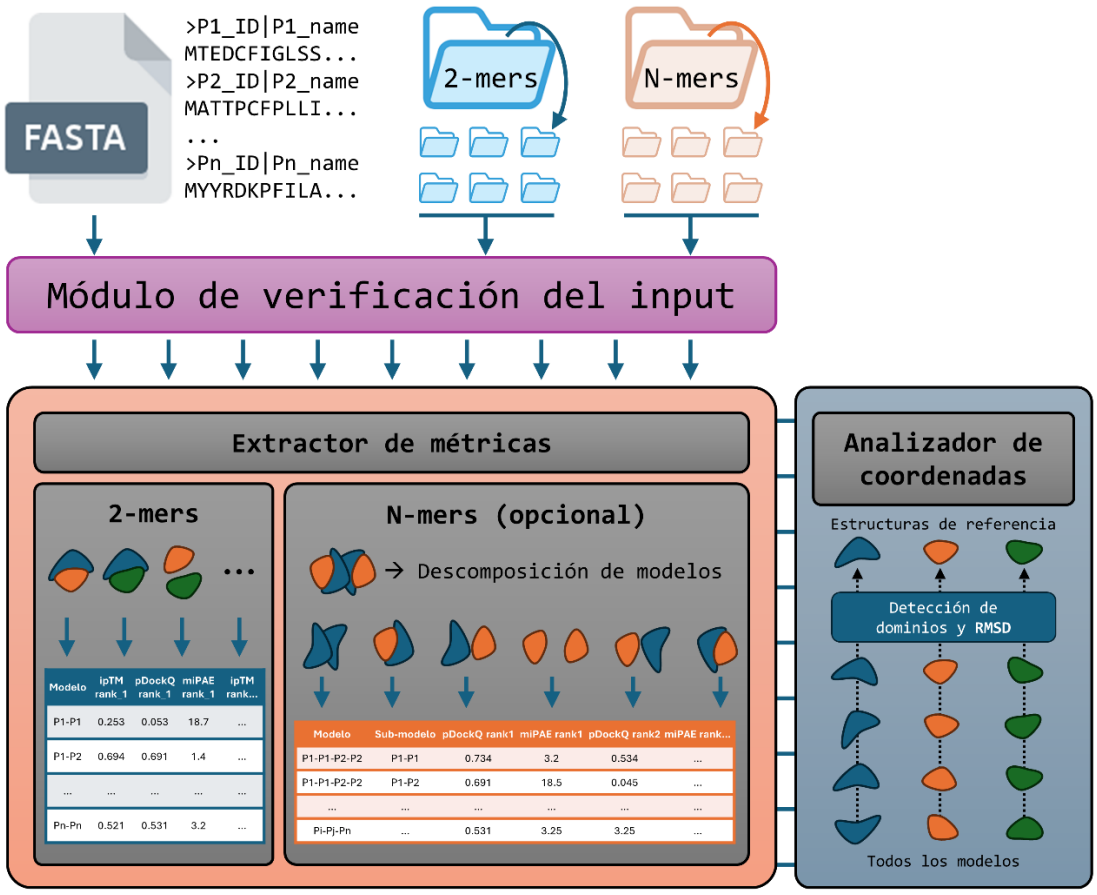

Fig. 22: *Input* de MultimerMapper, módulo de verificació de extracción de métricas y analizador de coordenadas. El input de MM consiste en un archivo FASTA con las secuencias, nombres e identificadores de las proteínas involucradas; y dos directorios con las predicciones de AF2m separadas (2-meros y N-meros). El directorio de N-meros es opcional. Un módulo de verificación corrobora que el formato de los archivos es el adecuado. El módulo siguiente extrae y computa métricas relevantes de los modelos. En el caso de los dímeros, las metricas pueden computarse directamente y se almacenan en forma tabular. Los N-meros son descompuestos en las combinaciones componentes para convertirlos en modelos diméricos y analizar sus superficies de interacción por separado. Esta información también se almacena en formato tabular. El analizador de coordenadas explora todas las conformaciones de cada proteína, elige una estructura de referencia, define dominios a partir de la matriz PAE y luego computa el RMSD por dominio de cada estructura contra la referencia.

El primer módulo de análisis de MM analiza los 2-meros y los N-meros por separado. Por un lado, los 2-meros son procesados directamente, extrayendo todas las métricas del modelo (pLDDT, pTM, ipTM y matriz PAE). Entre las métricas más importantes que se computan, la dimensionalidad de la matriz PAE se reduce convirtiéndola al miPAE y se calcula el pDockQ. Además, se calculan otras métricas que formaran parte

del *output* de MM (pLDDT promedio, pLDDT por proteína, etc.). Todos estos valores son generados para cada uno de los 5 modelos predicho por AF2m (rank1, rank2, …, rank5) y son cargados a la memoria en formato tabular utilizando “Pandas”. Por otro lado, cada N-mero es descompuesto en sus pares componentes. Por ejemplo, un 4-mero formado por 2 subunidades de A y dos subunidades de B ($A_1A_2B_1B_2$) es descompuesto en todas sus combinaciones diméricas posibles (A_1A_2, A_1B_1, …, B_1B_2). Así, es posible computar las métricas mencionadas sobre cada sub-PDB resultante y su sub-matriz PAE asociada. Nuevamente, la información es almacenada en formato tabular.

En paralelo, los modelos son procesados por un módulo que analiza las coordenadas atómicas y las submatrices PAE de las proteínas individuales. En primer lugar, se extrae una estructura de referencia para cada proteína extrayendo el monómero con el mejor pLDDT promedio. Posteriormente, utiliza un algoritmo de clusterización de la matriz PAE ampliamente utilizado para predecir dominios (https://github.com/tristanic/pae_to_domains) que fue modificado para aplicarse de forma interactiva o automática. En el caso en que se elija la opción interactiva, el programa genera una salida en formato HTML que se reproduce automáticamente en el navegador del sistema operativo, donde de muestra el *backbone* 3D de las proteínas, los dominios detectados y los valores de pLDDT por residuo. El usuario puede modificar los parámetros del algoritmo de clusterización y observar cómo cambian los límites de los dominios en tiempo real hasta encontrar la mejor combinación de parámetros que mejor describen los dominios globulares de las proteínas. Dado que MM está pensado para ejecutarse de forma repetida, es posible guardar las definiciones de dominios generadas para evitar la detección interactiva en la siguiente ejecución. En el caso de elegir la detección automática, se aplicará la misma combinación de parámetros por defecto o personalizados a todos los modelos. Una vez detectados los dominios, el algoritmo identifica que dominios son globulares y cuáles son los segmentos desordenados (pLDDT promedio por dominio < 60, por defecto). Sobre los dominios globulares, se calcula el valor RMSD (*Root Mean Square Deviation*) de todos los modelos monoméricos contra la referencia. Esto permitirá identificar más tarde que dominios sufren cambios conformacionales al ser modelados en distintos contextos. Vale aclarar que los dominios detectados no necesariamente

corresponden al concepto clásico de dominio proteico (unidad funcional con estructura y perfil de secuencia conservado evolutivamente), sino que corresponden a segmentos continuos de cada proteína que comparten valores de PAE contiguos similares (vea los dominios de la **Fig. 5**).

El flujo y procesamiento posterior de la información obtenida en esta primera etapa se muestra en la **Fig. 23**. Por un lado, las tablas generadas por el extractor de métricas son analizadas por un módulo detector de PPIs que utiliza las métricas de corte descriptas previamente (**Fig. 21** y **Fig. 22**) para identificar pares de proteínas que interactúan dentro de cada modelo. Esta información es compartida con un módulo de extracción de contactos y con un módulo de conversión a grafos de PPIs. El conversor se encarga de transformar la información de interacciones directas a grafos utilizando el paquete “igraph”, tal como se describió en las secciones anteriores (**Fig. 11** y **Fig. 14**). El resultado principal es una representación computacional de las redes de interacciones directas que puede ser acoplado a otros *pipelines* y una representación visual interactiva en formato HTML generada con el paquete “Plotly” que puede ser ejecutada en cualquier navegador. El grafo contiene múltiples metadatos que pueden ser visualizados a través de menús desplegables que proporcionan información sobre las proteínas (dominios, regiones desordenadas, cambios conformacionales, etc.) y sobre las interacciones (dinamismo, Nº de modelos que la contienen, contexto, etc.).

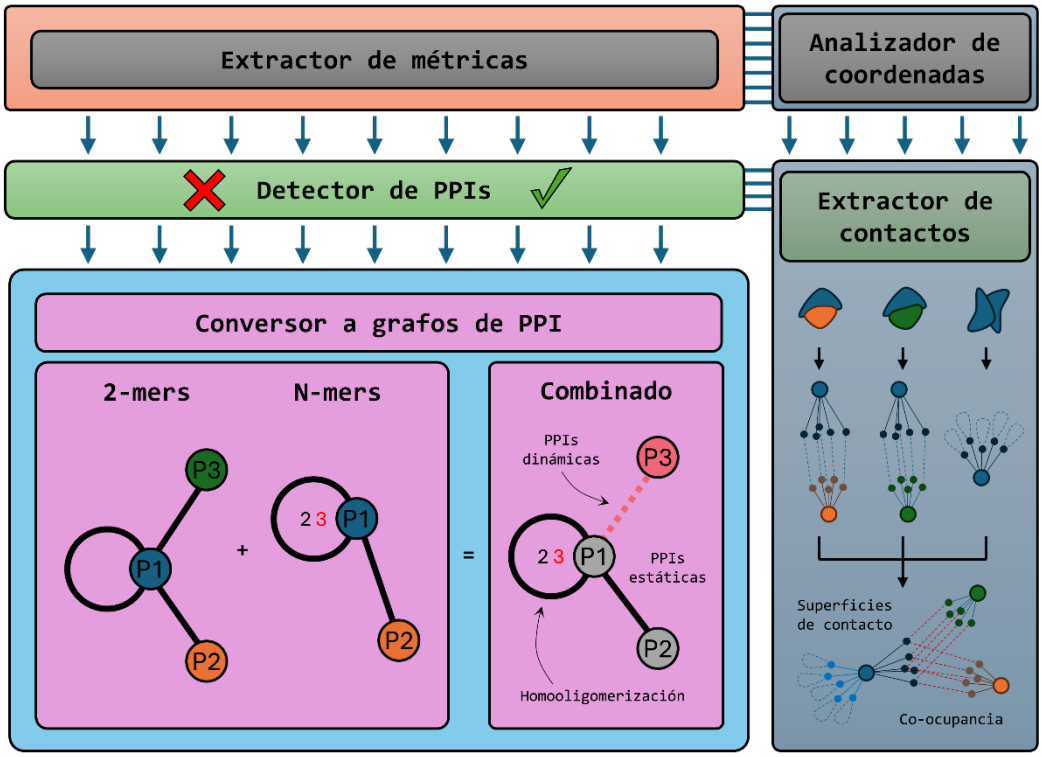

Fig. 23: Módulos de detección de PPIs, de conversión a grafos de PPIs y extractor de contactos. Las salidas tabulares del extractor de métricas son la entrada del módulo de detección de PPIs, que detecta pares de proteínas interactoras dentro de cada modelo. Esta información es compartida con el módulo de conversión a grafos de PPIs, que combina la información de interacción de los 2-meros y los N-meros para detectar PPIs dinámicas y los estados de homooligomerización de las proteínas. La información también se comparte con el módulo extractor de contactos que analiza las superficies de interacción entre las proteínas y convierte las coordenadas atómicas en grafos de contactos residuo-residuo que permite identificar superficies co-ocupadas por más de una proteína.

Algo que se anticipó brevemente al describir la lógica de la conversión a grafos de PPIs (vea la sección “Los problemas del ensamblado combinatorio, la hipótesis de las interacciones dinámicas y una posible solución”) es que la interpretación de las interacciones dentro de los N-meros deberían ser consideradas como un espectro, más que como un estado binario (dinámico/no dinámico). Para ello se generó un sistema heurístico (**Tabla 1**) que permite clasificar las interacciones desde las “más estáticas” (las que siempre aparecen, sin importar el contexto de modelado) hasta las “más dinámicas” (las que siempre tienden a desaparecer al introducir otras proteínas en el modelo), o incluso detectar las interacciones indirectas para poder descartarlas (interacciones mediadas a través de una proteína intermedia).

Tabla 1: Sistema de clasificación del espectro de interacciones implementado en MultimerMapper. El tipo de interacciones entre pares de proteínas se obtiene comparando las aristas (interacciones) del grafo de PPIs que utiliza solo los 2-meros con el grafo que utiliza únicamente los N-meros. La columna de 2-meros indica si la arista está presente o ausente en el grafo PPI de 2-meros y la columna de N-meros si está presente en el grafo de PPI de N-meros. La variación en N-meros indica el porcentaje de los modelos de N-meros que contienen el par de proteínas y que también contienen la arista. La columna N-meros pDockQ muestra los valores promedio de pDockQ de los modelos que superan el cutoff de miPAE. Este valor solo se aplica para distinguir interacciones directas e indirectas cuando la PPI está ausente en el grafo de 2-meros.

Tipo de interacción	2-meros	N-meros	Variación en N-meros	N-meros pDockQ
Estática	Presente	Presente	100% presente	No aplica
Predominantemente Estática	Presente	Presente	≥50% presente	No aplica
Dinámica positiva	Ausente	Presente	100% presente	>0.23
Predominantemente Dinámica positiva	Ausente	Presente	≥50% presente	>0.23
Dinámica Positiva Débil	Ausente	Presente	<50% presente	>0.23
Dinámica Negativa Débil	Presente	Presente	<50% presente	No aplica
Dinámica Negativa	Presente	Ausente	0% presente	No aplica
Indirecta	Ausente	Presente	> 0% presente	<0.23

Por la otra vía, el módulo de extracción de contactos, como su nombre lo indica, se encarga de extraer contactos residuo-residuo para obtener información extra de las interfases de interacción. El algoritmo considera como contactos a los pares de residuos de distintas proteínas que estén lo suficientemente cerca en el espacio (distancia intercentroide < 8 Å) y con buenas métricas (pLDDT>50 en ambos residuos, PAE < 9 Å). Posteriormente, agrupa las matrices de contacto y convierte las coordenadas de los centroides de las distintas superficies de contacto identificadas a nubes de punto, conectándolos al centro de masa de la proteína y al correspondiente contacto, generando un grafo con representación 3D en formato HTML interactivo. Entre otras cosas, estas representaciones permiten identificar rápidamente cuantas superficies de interacción posee cada proteína a través de códigos de colores, cuáles son las regiones de homodimerización y si existen superficies co-ocupadas por distintas proteínas. Este módulo representa la red, las proteínas y sus propiedades utilizando programación orientada al objeto mediante las clases de MM llamadas `Residue`, `Surface`, `PPI`, `Protein` y `Network`.

Además, la librería MM posee múltiples funciones de uso interactivo que permiten extraer información extra del conjunto de modelos. Por ejemplo, la función

"`RMSD_trajectory`" permite crear trayectorias de segmentos de proteínas ordenando los modelos según el RMSD de dicho segmento con respecto a una referencia, con múltiples parámetros personalizables. Esta permite capturar la movilidad del segmento deseado en función del modelo, para capturar cambios conformacionales en forma de trayectorias. Otra función importante es "`generate_filesystem_for_CombFold`", que genera el sistema de archivos necesarios para ejecutar el algoritmo combinatorio con las estequiometrías que se infieren utilizando los *outputs* de MM.

Ejemplos de ejecución de MultimerMapper sobre *datasets* reales

MM puede ejecutarse dentro de cualquier IDE que permita ejecutar Python 3.11 de forma interactiva utilizando los módulos, funciones y clases del paquete. El paquete puede importarse utilizando el alias `mm`, de la siguiente manera:

```
# Carga del paquete
import multimer_mapper as mm
```

En este caso, utilizaremos como ejemplo un *dataset* de un complejo nuclear de *Trypanosoma brucei* relacionado con la acetilación de histonas: el complejo NuA4. Las listas de proteínas utilizadas (**Tabla 2**) para predecir las combinaciones con AF2m (2-meros y N-meros) provienen de experimentos de coinmunoprecipitación (CoIP) de distintas proteínas del complejo que fueron identificadas utilizando espectrometría de masas (Staneva, 2021).

Tabla 2: Listas de proteínas recuperadas en experimentos de CoIP del complejo NuA4 de *T. brucei*.

ID	Nombre	Símbolo
Tb927.1.3400	Bromodomain factor 6	BDF6
Tb927.7.4560	Histone acetyltransferase 1	HAT1
Tb927.10.8310	Histone acetyltransferase 3	HAT3
Tb927.7.5310	YEATS domain protein 2	YEA2
Tb927.8.5320	MRG binding protein	MRGBP
Tb927.9.2910	Esa1-associated factor 6	EAF6
Tb927.11.7880	Yeast ING1 homolog 2	YNG2
Tb927.11.3430	YNG2-like	YNG2L
Tb927.10.14190	Enhancer of Polycomb Like 1	EPL1
Tb927.1.650	MORF4 related gene x	MRGx
Tb927.10.9930	PHD finger domain protein 1	PHD1
Tb927.6.1240	Hypothetical protein	Tb927.6.1240
Tb927.9.12900	RNA polymerase-associated protein LEO1	LEO1

Debemos definir el directorio de trabajo que contiene todo el sistema de archivos que utilizaremos como *input*, indicando el camino donde se encuentra el archivo FASTA, los caminos a las predicciones (2-meros y N-meros) y una carpeta de salida:

```
# Elegimos el directorio de trabajo
import os
os.chdir("C:/Users/elvio/Desktop/NuA4")

# Definimos el camino de nuestro archivo FASTA
fasta_file_path = "./NuA4_proteins.fasta"

# Caminos de los directorios conteniendo las predicciones de AF2m
AF2_2mers = "./2-mers"
AF2_Nmers = "./N-mers"

# Camino para guardar los archivos de salida
out_path = "./mm_output"
```

Si bien la mayoría de los parámetros de MM se encuentran definidos por defecto en un módulo de configuración por defecto (cfg/default_settings.py), es posible modificarlos dentro del ambiente de trabajo. Algunos pueden afectar los resultados de interacciones, como los valores de *cutoffs* para diferentes FPR (**Tabla Suplementaria 1**); otros solo afectarán la estética de las visualizaciones (como paletas de colores) y otros permitirán guardar los resultados de ejecuciones previas, como las definiciones de dominios del modo interactivo. Algunos de los parámetros más importantes se definen a continuación:

```
# Si se desea trabajar con nombres, utilizar True (False: se usarán IDs en las visualizaciones)
use_names = True

######################### Parámetros del algoritmo de detección de dominios ############################

# Resolución de clusterización (0.075 por defecto)
graph_resolution = 0.075

# Para usar la detección de dominios automática: True. False: modo interactivo.
auto_domain_detection = False

# Al rehacer todo el pipeline, es mejor guardar el preset generado durante el modo interactivo.
# La primera usar save_preset=True, lo que guarda el preset en la carpeta "./domains"
# como "{fasta_file_name}-graph_resolution_preset.json".
save_preset = False
# Las siguientes veces que se ejecute MM, usar save_preset=False, definiendo
# graph_resolution_preset = "./domains/{fasta_file_name}-graph_resolution_preset.json"
graph_resolution_preset = None

# Cutoff para considerar un dominio desordenado (domain_mean_pLDDT < cutoff => desordenado)
domain_RMSD_plddt_cutoff = 60
# Residuos con pLDDT menor a trimming_RMSD_plddt_cutoff serán ignorados durante los calculos
trimming_RMSD_plddt_cutoff = 70


############ Valores de corte del detector de PPIs (Ver Tabla Suplementaria 1) ########################

# Para detectar interacciones en 2-meros
min_PAE_cutoff_2mers = 6.16
ipTM_cutoff_2mers = 0.24

# Para detectar interacciones en N-meros
```

```
min_PAE_cutoff_Nmers = 6.16
pDockQ_cutoff_Nmers = 0.051   # As ipTM loses sense in N-mers, we use pDockQ with a low cutoff value

# Cutoffs generales
N_models_cutoff = 3
pdockq_indirect_interaction_cutoff = 0.23
predominantly_static_cutoff = 0.6


################ Paleta de colores amigable para daltónicos (Paul Tol's + orange) ######################

PT_palette = {
    "black"          : "#000000",
    "green"          : "#228833",
    "blue"           : "#4477AA",
    "cyan"           : "#66CCEE",
    "yellow"         : "#CCBB44",
    "purple"         : "#AA3377",
    "orange"         : "#e69f00",
    "deep orange"    : "#d55e00",
    "red"            : "#EE6677",
    "gray"           : "#bbbbbb",
    }
```

Para ejecutar el extractor de métricas, detectar interacciones y construir los grafos de PPI (vea **Fig. 22** y **Fig. 23**), se utiliza la función `parse_AF2_and_sequences`, la cual genera el objeto `mm_output`, un diccionario que guarda todos los objetos generados por MM y que puede utilizarse para combinar con otros *pipelines*. La forma más simple de ejecución requiere solo el archivo FASTA, los 2-meros y una carpeta de salida:

```
# Ejecutar el extractor de datos (forma simple)
mm_output = mm.parse_AF2_and_sequences(fasta_file_path, AF2_2mers, out_path)
```

Esto extraerá todas las métricas de los modelos, aplicará el modelo de detección de PPIs y construirá las representaciones computacionales de los grafos de PPIs utilizando los parámetros por defecto de MM. Para modificar algunos de los parámetros:

```
# Ejecutar el extractor de datos (forma personalizada)
mm_output = mm.parse_AF2_and_sequences(
            # Input
            fasta_file_path, AF2_2mers, AF2_Nmers, out_path,
            # Usar nombres en vez de IDs
            use_names = True,

            # ----------- Opciones del detector de dominios ----------
            # Definir dominios de forma interactiva
            graph_resolution = graph_resolution, auto_domain_detection = False,
            # Mostrar las estructuras y submatrices PAE en tiempo real
            show_structures = True, display_PAE_domains = True,
            # Guardar el preset
            save_preset = True,
            # Guardar los resultados de cada dominio
            save_PAE_png=True, save_domains_html=False, save_domains_tsv=False,

            # ----------- Opciones de detección de PPIs ----------
            # Cutoffs de 2-meros (ver Tabla Suplementaria 1)
            min_PAE_cutoff_2mers = min_PAE_cutoff_2mers,
            ipTM_cutoff_2mers = ipTM_cutoff_2mers,
            # Cutoffs de N-meros (ver Tabla Suplementaria 1)
            min_PAE_cutoff_Nmers = min_PAE_cutoff_Nmers,
```

```
            pDockQ_cutoff_Nmers = pDockQ_cutoff_Nmers,
            # Cutoffs generales
            N_models_cutoff = N_models_cutoff,
            pdockq_indirect_interaction_cutoff = \
            pdockq_indirect_interaction_cutoff,
            predominantly_static_cutoff = predominantly_static_cutoff,

            # ----------- Opciones de paletas de colores en grafos ----------
            edge_color1=PT_palette["red"], edge_color2=PT_palette["green"],
            edge_color3=PT_palette["orange"],
            edge_color4=PT_palette["cyan"], edge_color5=PT_palette["yellow"],
            edge_color6=PT_palette["blue"],
            edge_color_both=PT_palette["black"],
            vertex_color1=PT_palette["red"], vertex_color2=PT_palette["green"],
            vertex_color3=PT_palette["orange"],
            vertex_color_both=PT_palette["gray"]
)
```

En este caso ejecutaremos el modo interactivo de detección de dominios, utilizaremos los cutoffs definidos más arriba y personalizaremos los colores del grafo de salida usando una paleta *color blind friendly*. Además, utilizaremos el *dataset* completo, es decir, incluiremos todos los N-meros que alcanzaron la convergencia. El tiempo de ejecución dependerá del tamaño del *dataset* y de la velocidad del procesador que dispongamos. Al ejecutarse, la consola informará el porcentaje de progreso de cada módulo y se generará un archivo con el *logging* del programa (multimer_mapper.log) con información sobre el progreso, advertencias o errores. En este caso, el *dataset* cuenta con 79 2-meros y 37 N-meros, demorando aproximadamente 40 minutos en finalizar utilizando una computadora portátil con CPU Ryzen 3700X y 16 GB de RAM, con picos de utilización de memoria RAM cercanos al 80%. La misma computadora utilizando solamente 8 GB de memoria RAM resultó en errores de memoria, por lo que sería recomendable disponer de al menos 32 GB de memoria para conjuntos más grandes.

Dado que elegimos el modo interactivo de detección de dominios, cuando la ejecución alcanza el módulo analizador de coordenadas, el programa se detiene esperando nuestra intervención. En este punto, se mostrará la matriz PAE del PDB de referencia de la primera proteína del *dataset* con los dominios resultantes y se abrirá una ventana en nuestro navegador con la estructura de referencia y sus dominios en códigos de colores (**Fig. 24**). Podremos explorar la estructura del monómero desde el navegador, con la posibilidad de cambiar el código de colores de la estructura para mostrar valores de pLDDT por residuo. El programa nos preguntará si estamos de acuerdo con los dominios detectados y podremos elegir si modificar (o no) el valor de resolución del algoritmo de clusterización. En el caso de modificarlo, se actualizará la ventana y se

nos preguntará nuevamente. El proceso se repetirá por cada proteína del dataset. Los resultados de este módulo se guardarán dentro del subdirectorio `domains`, que se creará automáticamente en la carpeta de salida.

Fig. 24: Modo de detección de dominios interactivo. (**A**) Resultados de la clusterización de la matriz PAE con las definiciones de dominios. (**B**) Gráfico interactivo 3D con código de colores por dominio. (**C**) Gráfico interactivo 3D seleccionando la opción de código de colores por pLDDT (rojo: bajo pLDDT; azul: alto pLDDT).

El diccionario `mm_output` generado contiene sub-diccionarios, *dataframes* de Pandas y grafos de igraph con la información extraída de contactos, proteínas, modelos, etc. Por ejemplo, la llave (*key*) "`sliced_PAE_and_pLDDTs`" es un sub-diccionario que contiene información sobre las proteínas individuales. Las sub-*keys* de este sub-diccionario serán los IDs o los nombres las proteínas, dependiendo si definimos `use_names` como `True` o `False`:

```
# Diccionario con información de cada proteína. Proviene de los 2-meros y de los N-meros
mm_output['sliced_PAE_and_pLDDTs'].keys()
```

```
Out[1]: dict_keys(['BDF6', 'HAT1', 'HAT3', 'YEA2', 'MRGBP', 'EAF6', 'YNG2', 'YNG2L', 'EPL1', 'MRGx', 'PHD1',
'Tb927.6.1240', 'LEO1'])
```

A su vez, cada sub-*key* contiene subdiccionarios con la información de secuencia, la longitud, los archivos PDBs que contienen a la proteína, los valores de pLDDT por residuo de cada PDB, información sobre la estructura de referencia de la proteína elegida, entre otros. Podemos explorar su contenido utilizando las sub-*keys:*

```
# Extraemos ID de la primera proteína del diccionario y vemos cuales son las sub-keys
protein0_ID = list(mm_output['sliced_PAE_and_pLDDTs'].keys())[0]
mm_output['sliced_PAE_and_pLDDTs'][protein0_ID].keys()
```

```
Out[1]: dict_keys(['sequence', 'length', 'PDB_file', 'pLDDTs', 'max_mean_pLDDT_index', 'best_PAE_matrix',
'PDB_xyz'])
```

```
# Algunas sub-keys útiles
mm_output['sliced_PAE_and_pLDDTs'][protein0_ID]["sequence"]       # Secuencia de aminoacidos
```

```
mm_output['sliced_PAE_and_pLDDTs'][protein0_ID]["length"]          # Longitud en aminoacidos
mm_output['sliced_PAE_and_pLDDTs'][protein0_ID]["PDB_file"]        # PDBs que contienen la proteina
mm_output['sliced_PAE_and_pLDDTs'][protein0_ID]["pLDDTs"]          # pLDDT por residuo de cada PDB
mm_output['sliced_PAE_and_pLDDTs'][protein0_ID]["best_PAE_matrix"] # Mejor matriz PAE
mm_output['sliced_PAE_and_pLDDTs'][protein0_ID]["PDB_xyz"]         # Objeto Bio.PDB.Chain.Chain
```

Entre los *dataframes* generados encontraremos:

- `domains_df`: Contiene las posiciones de inicio y fin de cada dominio de cada proteína. También contiene los valores de pLDDT promedio del dominio en la estructura de referencia.
- `pairwise_2mers_df`: Contiene las métricas de todos los modelos de 2-meros.
- `pairwise_2mers_df_F3`: Contiene las métricas de los modelos de 2-meros que superaron los *cutoffs* del módulo de detección de PPIs.
- `pairwise_Nmers_df`: contiene las métricas de todos los pares de proteínas dentro de cada uno de los modelos de N-meros.
- `pairwise_Nmers_df_F3`: Contiene las métricas de los pares de proteínas dentro de los modelos de N-meros que superaron los *cutoffs* del módulo de detección de PPIs.

```
# Dataframes generados:
mm_output['domains_df']             # Dominios detectados
mm_output['pairwise_2mers_df']      # Métricas de todos los pares dentro de los 2-meros
mm_output['pairwise_2mers_df_F3']   # Métricas los pares que superaron cutoffs dentro de los 2-meros
mm_output['pairwise_Nmers_df']      # Métricas de todos los pares dentro de cada N-meros descompuesto
mm_output['pairwise_Nmers_df_F3']   # Métricas los pares que superaron cutoffs dentro de los N-meros
```

Entre los objetos que representan grafos de igraph, encontraremos:

- `graph_2mers`: Contiene la representación computacional del grafo generado a partir de los 2-meros.
- `graph_Nmers`: Contiene la representación computacional del grafo generado a partir de los N-meros.
- `combined_graph`: Contiene la representación computacional del grafo generado a partir de la comparación entre el grafo de 2-meros y el de los N-meros.
- `fully_connected_subgraphs`: Cuando existan subconjuntos de proteínas que no se conecten, contendrá una lista de los subgrafos que estos conjuntos forman.

```
# Graphs data
mm_output['graph_2mers']
mm_output['graph_Nmers']
mm_output['combined_graph']
mm_output['fully_connected_subgraphs']
```

Si bien la función genera algunas representaciones visuales de los grafos que se guardan en el directorio `2D_graphs` dentro de la carpeta de salida (**Fig. 25**), estas no son interactivas. Mientras tanto, los objetos del tipo grafo son meras representaciones computacionales sobre las cuales se puede continuar operando dentro de Python, permitiendo combinarlos con otros *pipelines*.

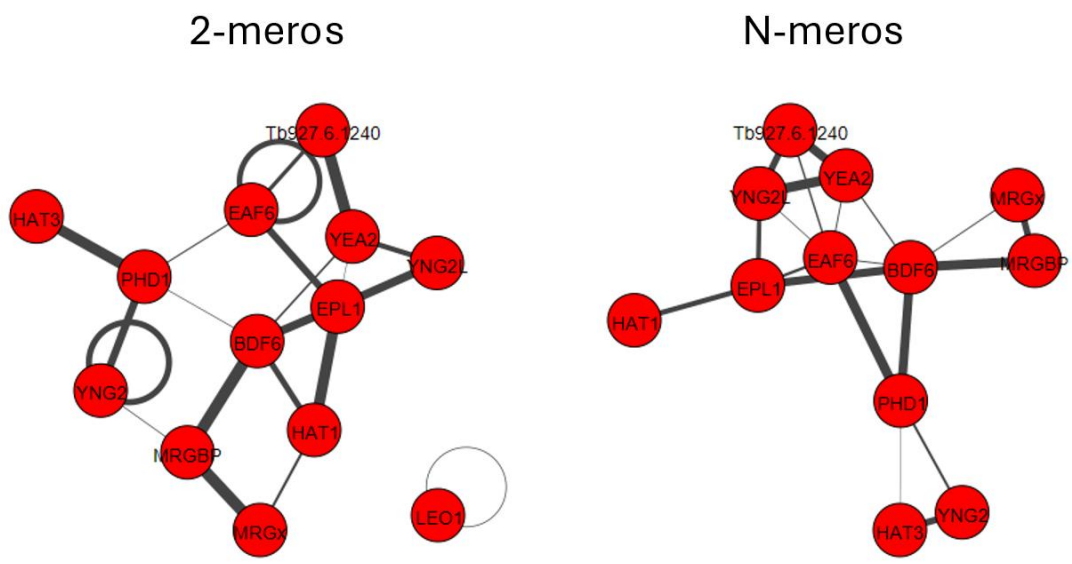

Fig. 25: Grafos no interactivos de 2-meros y N-meros. Note como algunas aristas o proteínas desaparecen en el grafo de N-meros.

El grafo que nos interesa es `combined_graph`. Para obtener una representación visual interactiva, es necesario primero generar un *layout* con alguno de los algoritmos proporcionados por el paquete igraph. Esto podemos hacerlo utilizando el método `layout`, de la siguiente manera:

```
# Crear un layout para el grafo combinado (usa todos los tipos de interacciones)
combined_graph_layout = mm_output['combined_graph'].layout("fr")
```

En este caso usaremos el algoritmo estocástico de Fruchterman-Reingold (“fr”) para encontrar la disposición óptima de los vértices, aunque existen muchos otros. Por ejemplo, el algoritmo determinístico de Kamada-Kawai (“kk”) o el algoritmo circular (“circle”), que simplemente distribuye los vértices en un círculo. Visite

https://igraph.org/python/tutorial/0.9.8/tutorial.html para obtener una lista extensiva de los algoritmos que pueden utilizarse. En el caso que desee construir representaciones interactivas solo con algún tipo de interacción, MM contiene la función de envoltorio `generate_layout_for_combined_graph`. Por ejemplo, para generar un *layout* que incluya solo las interacciones estáticas usando el algoritmo “kk”:

```
# Crear layout usando solo las aristas que se desee
combined_graph_layout_2 = MM.generate_layout_for_combined_graph(
    combined_graph,
    # You can choose which type of interactions use to built the layout
    edge_attribute_value=['Static interaction',
                          'Predominantly static interaction'],
    layout_algorithm="kk")
```

Crearemos la visualización interactiva utilizando el primer *layout* con la función de MM `igraph_to_plotly`, de la siguiente manera:

```
# Conversión de objeto igraph a grafo interactivo
combined_graph_interactive = mm.igraph_to_plotly(

            # Grafo combinado y layout elegido (parámetros posicionales obligatorios)
            mm_output['combined_graph'], combined_graph_layout,

            # Modificar la orientación de las aristas de homooligomerización
            # (0: sin cambio, 0.25: un cuarto de vuelta, etc.)
            self_loop_orientation = 0.25,
            # Modificar el tamaño de las aristas de homooligomerización
            self_loop_size=4,

            # Quitar los ejes, el background y mostrar los nombres de proteínas en negrita
            show_axis= False, showgrid= False, use_bold_protein_names= True,

            # Resaltar dominios con RMSD mayor que este valor en los modelos
            add_bold_RMSD_cutoff = 5,

            # Se puede guardar la salida en formato HTML para compartirlo (incluso por celular)
            save_html = 'NuA4_2D_graph.html',

            # Esta opción es altamente recomendable para saber que cutoffs se usaron al realizar pruebas
            add_cutoff_legend = True)
```

Al ejecutar este comando, se abrirá una ventana nueva en nuestro navegador con el grafo interactivo (**Fig. 26**). Además, se guardará el archivo HTML con el nombre que elegimos usando la opción `save_html`, el cual podremos compartir con colegas y ejecutar en cualquier navegador. Los colores de las aristas (interacciones) responden a las clasificaciones de la **Tabla 1** y la leyenda indica su significado. El tamaño de las aristas estáticas es directamente proporcional a la intensidad de la interacción, es decir, mientras mejores sean las métricas de interacción promedio, más gruesas las líneas. El color de proteínas (vértices) también indican la clasificación de la proteína en el grafo según su comportamiento. Por ejemplo, la proteína LEO1 y sus interacciones desaparecen en el grafo de N-meros (vea la **Fig. 25**), por lo que es

clasificada como dinámica y tanto vértice como sus interacciones aparecen en color rojo. La elección de los colores y la textura de las aristas está pensada para que las interacciones más estáticas sean más oscuras y sólidas, mientras que las más dinámicas sean más claras y discontinuas.

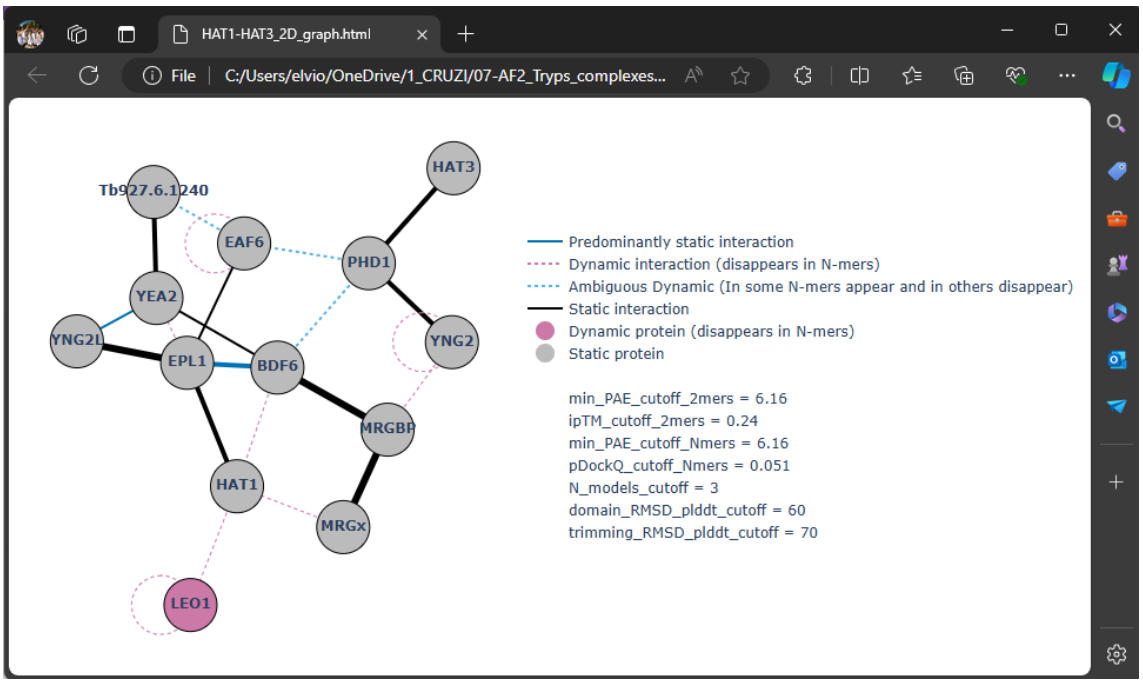

Fig. 26: Grafo de PPIs interactivo del *dataset* de NuA4. La leyenda contiene los significados de los colores de las interacciones (aristas) y de las proteínas (vértices).

A su vez, si desplazamos el cursor sobre las proteínas o de las interacciones se desplegará una ventana de forma interactiva que nos proporcionará información adicional. En la **Fig. 27** se muestra las propiedades de la proteína EPL1. En primer lugar, veremos la clasificación del vértice dentro del grafo, en este caso una proteína estática (está presente en ambos grafos), y su identificador único (ID). Le sigue una tabla con los límites de los dominios detectados y sus valores de pLDDT promedio extraídos de la estructura de referencia. Por ejemplo, el primer dominio es una región desordenada (tiene muy bajo pLDDT promedio) y el segundo dominio sería un dominio globular (alto pLDDT promedio). Por último, se muestran los valores de pLDDT promedio de los dominios ordenados extraídos de cada una de las cadenas de los modelos que contienen a la proteína, junto a los valores de RMSD calculados entre el dominio de esa cadena y el mismo dominio de la estructura de referencia. En este ejemplo, podemos ver como el dominio 2 de la proteína EPL1 obtiene valores más

elevados de pLDDT promedio al estar presente HAT1 en los modelos. A su vez, lo valores de RMSD son menores. Esto indicaría que este dominio se estabiliza al estar presente HAT1, por lo que podremos ir a ver las estructuras para explicar el mecanismo (**Fig. 28**).

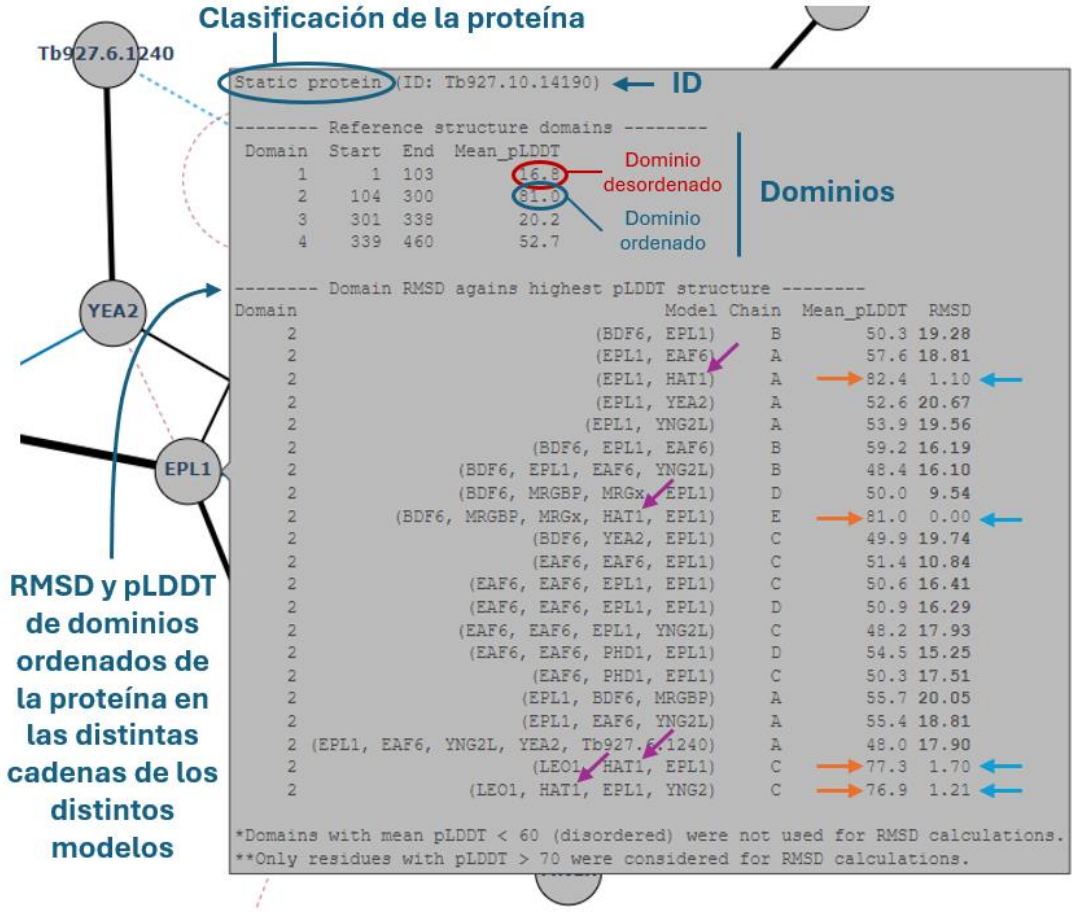

Fig. 27: Ejemplo de propiedades de la proteína (vértice) EPL1. La clasificación de la proteína en el grafo combinado aparece arriba, seguida de su identificador único (ID). Los dominios identificados, sus posiciones de comienzo/fin y el valor de pLDDT promedio se muestran en el medio. Por último, se muestra una tabla con los valores de pLDDT promedio de los dominios ordenados y el valor de RMSD de cada cadena de los diferentes modelos. El RMSD es calculado contra el mismo dominio de la estructura de referencia. En este caso, solo el dominio 2 es considerado ordenado. Note que los modelos que contienen a HAT1 (flechas violetas) poseen valores de pLDDT promedio altos (flechas naranjas) y valores de RMSD bajos (flechas azules). Esto indica que el dominio 2 de EPL1 está sufriendo un cambio o estabilización conformacional cuando HAT1 está presente.

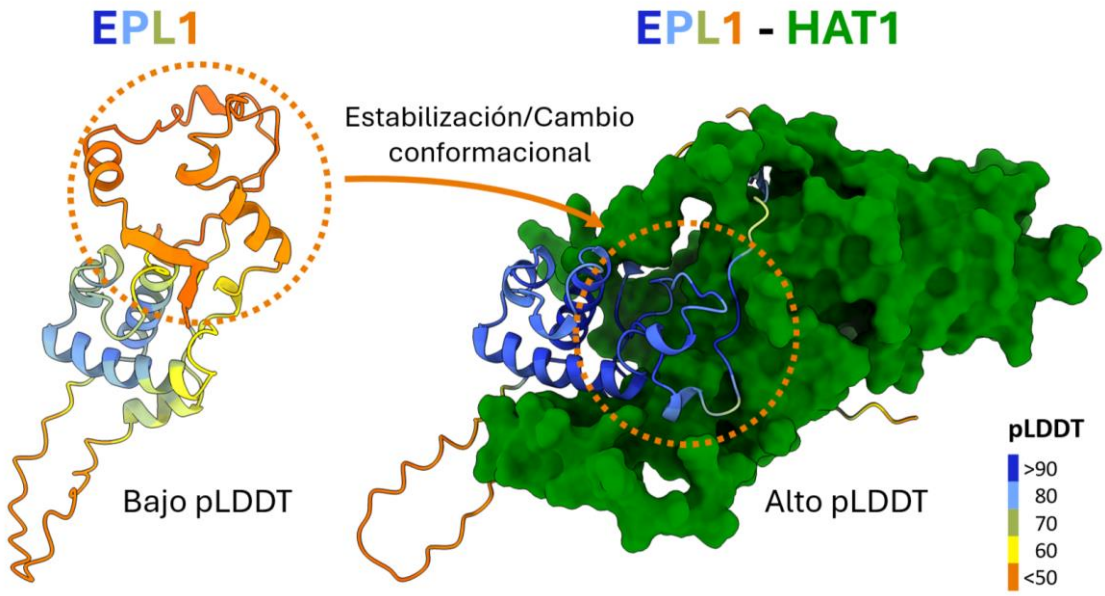

Fig. 28: Posible mecanismo de estabilización conformacional de EPL1 al interactuar con HAT1. La región de EPL1 dentro del círculo se estabiliza al interactuar con HAT1. Esto explicaría el incremento en RMSD en los modelos en los que HAT1 no está presente.

Por otra parte, en la **Fig. 29A** vemos como ejemplo las propiedades de la homointeracción de EAF6 observadas en el menú desplegable de la arista EAF6-EAF6. Este es un caso de interacción dinámica negativa total, es decir, la interacción está presente en los 2-meros, pero desaparece siempre que se introducen otras proteínas al modelo. En el menú podremos ver que el Nº de modelos que superan el cutoff (`N_models`) es de 5 para el 2-mero, mientras que casi ningún modelo de los N-meros supera los cutoffs. Para obtener más información del mecanismo, tendremos que generar el gráfico de contactos residuo-residuo de MM (ver más abajo). La **Fig. 29B** corresponde a un ejemplo de interacción estática total entre MRGx y MRGBP. En otras palabras, no importa que otras proteínas estén presentes en los modelos, la interacción MRGx-MRGBP siempre está presente. Así, podremos explorar las distintas interacciones directas presentes en nuestro *dataset* de estructuras, obteniendo información sin necesidad de explorar los modelos y sus métricas uno a uno. Además, tendremos una idea de cuáles son las interacciones centrales del complejo que estamos estudiando, las cuales se manifestarán como interacciones estáticas (líneas negras sólidas).

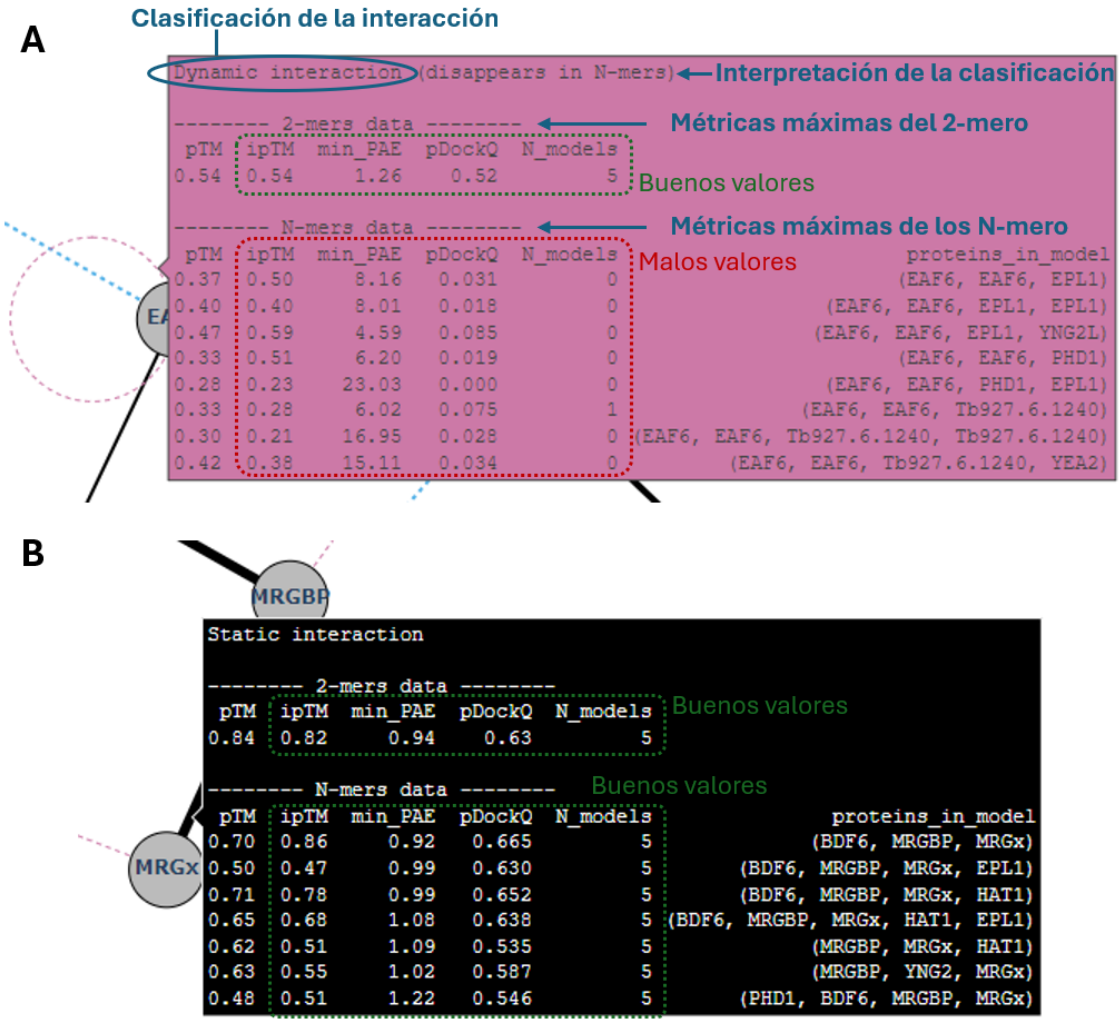

Fig. 29: Ejemplo de propiedades de las interacciones (aristas) de EAF6-EAF6 y MRGx-MRGBP. (**A**) Ejemplo de arista dinámica de homointeracción de EAF6. La dimerización se da solo cuando EAF6 es modelada en los 2-meros, mientras que desaparece al estar presentes otras proteínas. Esto se refleja en los valores de las métricas de los modelos. (**B**) Ejemplo de interacción estática entre MRGx y MRGBP. Ambos 2-meros y N-meros tienen buenas métricas.

Para generar los grafos 3D de contactos residuo-residuo (vea el módulo extractor de contactos de la **Fig. 23**), es necesario convertir el grafo combinado (`combined_graph`) que es un objeto de tipo `igraph.Graph` a un objeto de clase `multimer_mapper.Network`:

```
# Convertir grafo de PPI a Network de MultimerMapper
nw = mm.Network(mm_output['combined_graph'])
```

Este nuevo objeto posee atributos y métodos específicos de MM que permiten operar implícitamente sobre objetos de clase `multimer_mapper.Residue` (representa

residuos de aminoácidos), `multimer_mapper.Surface` (representa conjuntos de residuos de aminoácidos que establecen superficies de contacto), `multimer_mapper.PPI` (representa formas de interacción entre dos proteínas) y `multimer_mapper.Protein` (representa las proteínas).

```
type(nw)
Out[1]: multimer_mapper.Network
```

Al igual que con la salida del extractor de métricas (`mm_output`), podremos combinar esta información con otros *pipelines*, con la diferencia de que los métodos de cada clase fueron diseñados para operar a alto nivel. Por ejemplo, podemos extraer los objetos de clase mm.`Protein` simplemente llamando al método `get_proteins` del objeto `Network`, extraer la primera proteína de la red utilizando notación de indexado (corchetes) y obtener su secuencia con el método `get_seq` del objeto `Protein`:

```
# Extraer lista de objetos Protein de la red
lista_de_proteinas = nw.get_proteins()

# Extraer la primera proteína con método de indexado
primera_proteina = lista_de_proteinas[0]

# Extraer su secuencia con el método de extracción de secuencia
secuencia_primera_proteina = primera_proteina.get_seq()

# Imprimir la secuencia
print(secuencia_primera_proteina)
Out[1]:
"MRHVDVLQSLMVRLVDPEEEEHDWLLISSLFYSLTKLMLPATRVKELYYHYLQVHGSRPAMEAACKDFSNKLSEKLRQRIESTTTTTTATTLAGIHRSSVSGTSAAVG
VLDVTKVSAGAAGGEPGIPVGVLSGSPTVATSVWDESGFCGREAAFRKLTSEQKARWHNLRRELISPDILIDLVQQFTLVDGAAVFLAPVVPSTVMEFNGVRSGPYVTV
IHQPLSLMCVKRRVLAARRDYELHKHQSGAYLPTGQNNVGVGSRKRERGNGGVTSRASAVSQPPHFSIATNSGEKGNVIRTLQELEQAVWHITANCVMFNAPESYYPYV
ARKFALACVAIIDDYCTQRIAGV"
```

Si bien se trata de un ejemplo trivial, también se crearon métodos para extraer los valores de pLDDT por residuo (`get_res_pLDDT`), extraer las coordenadas atómicas 3D de cada residuo (`get_res_centroids_xyz`), mover linealmente las proteínas en el espacio (`translate`) o rotarlas sobre un eje (`rotate`), entre otros. Todos fueron diseñados para operar a alto nivel, es decir, con sintaxis lo más simple y transparente posible. Por lo tanto, todos los métodos que se aplican sobre el objeto de clase `Network`, bajo la cortina ejecutan estos métodos.

De forma análoga a como generamos un *layout* bidimensional para el grafo 2D, es necesario generar un *layout* 3D para el grafo de contactos, que luego se puede convertir a visualizaciones interactivas de plotly y de py3Dmol. Esto lo haremos

ejecutando los métodos `generate_layout` y 2 métodos de visualización, de la siguiente manera:

```
# Generar layout para el grafo de contactos
nw.generate_layout()

# Producir visualizaciones 3D con plotly y con py3Dmol
nw.generate_plotly_3d_plot(save_path = out_path + '/3D_graph_plotly.html')
nw.generate_py3dmol_plot(save_path = out_path + '/3D_graph_py3dmol.html')
```

El primer método distribuye las proteínas en el espacio a través de los métodos de translación y rotación de proteínas, generando una distribución que intenta minimizar el número de veces que las líneas de contacto se entrecruzan. Por defecto, se aplica un algoritmo estocástico que utiliza campos de fuerza y generará un *layout* diferente cada vez que ejecutemos el método. Por lo tanto, si no estamos conforme con la distribución de las proteínas, podemos ejecutarlo nuevamente para obtener una nueva distribución. Al ejecutar los 2 métodos siguientes, se abrirá una ventana en nuestro navegador con el grafo 3D de contactos utilizando plotly (**Fig. 30**) y py3Dmol (no se muestra). Algunas posibilidades interactivas que nos brinda el primer gráfico son:

- Manteniendo el botón izquierdo del *mouse* podremos rotar el grafo en 360º.
- Manteniendo el botón derecho del *mouse* desplazaremos el grafo sobre el plano de la pantalla.
- Con la rueda del *mouse* podremos hacer zoom.
- Los menús de la leyenda son interactivos, pudiendo activar o desactivar visualizaciones al hacer *click*. Por ejemplo, se puede visualizar el *backbone* de las proteínas, coloreados por pLDDT o por dominio.
- Al pasar por encima de un contacto o residuo, veremos información sobre los aminoácidos involucrados.

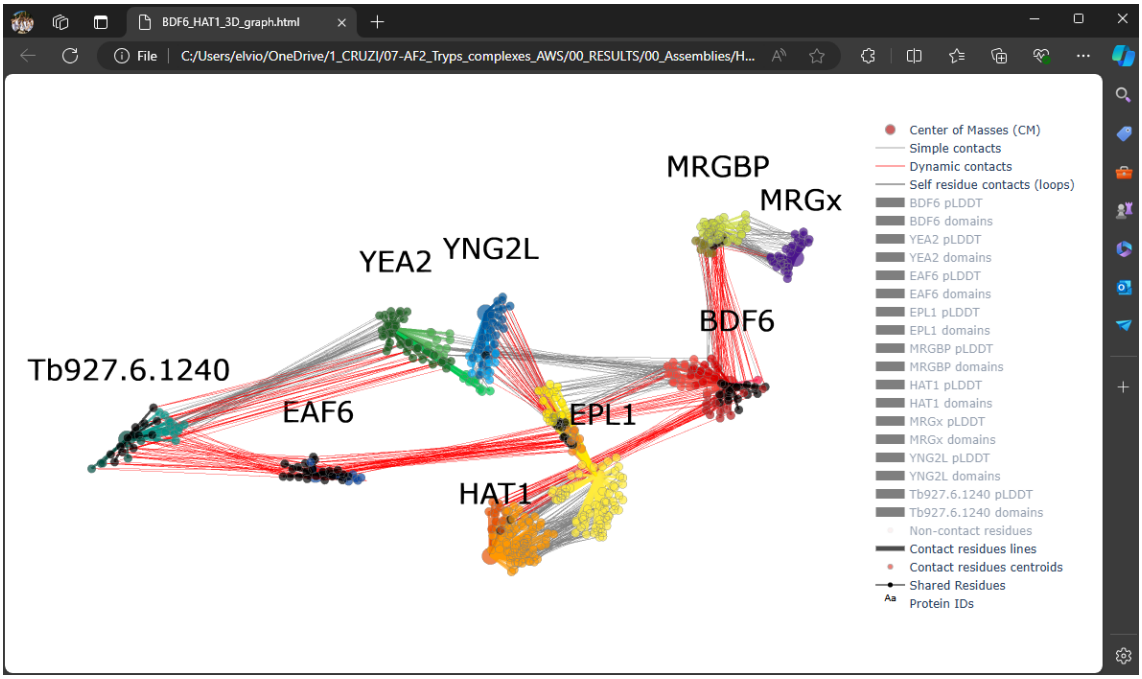

Fig. 30: Grafo 3D de contactos residuo-residuo. El menú izquierdo permite agregar o quitar capas de visualización al grafico de forma interactiva. Cada proteína está coloreada de una gama de colores distintos y los puntos representan los residuos que se encuentran en contacto. Las distintas superficies de interacción se muestran en distintas gamas de colores (por ejemplo, YEA2 posee 2 superficies, una verde claro y otra verde oscuro). Los residuos resaltados en negro son residuos involucrados en superficies de contacto co-ocupadas por distintas proteínas, y sus contactos con otros residuos se representan con líneas rojas. El resto de los contactos se representan en gris.

Retomemos que es lo que sucede con la homooligomerización de EAF6 analizando los datos del grafo 3D de contactos (**Fig. 31**). En un mismo gráfico, podemos ver de forma resumida que EAF6 interacciona consigo misma, con Tb927.6.1240 y con EPL1, todo a través de la misma superficie del dominio 1. Por lo tanto, al comparar esta información con lo obtenido a través del grafo combinado de PPIs (**Fig. 26** y **Fig. 29A**), deducimos que las interacciones EAF6-EAF6, EAF6-Tb927.6.1240 y EAF6-EPL1 no pueden coexistir porque se dan a través de la misma superficie de interacción, mientras que la única que prevalece en todas las condiciones de modelado es la interacción EAF6-EPL1.

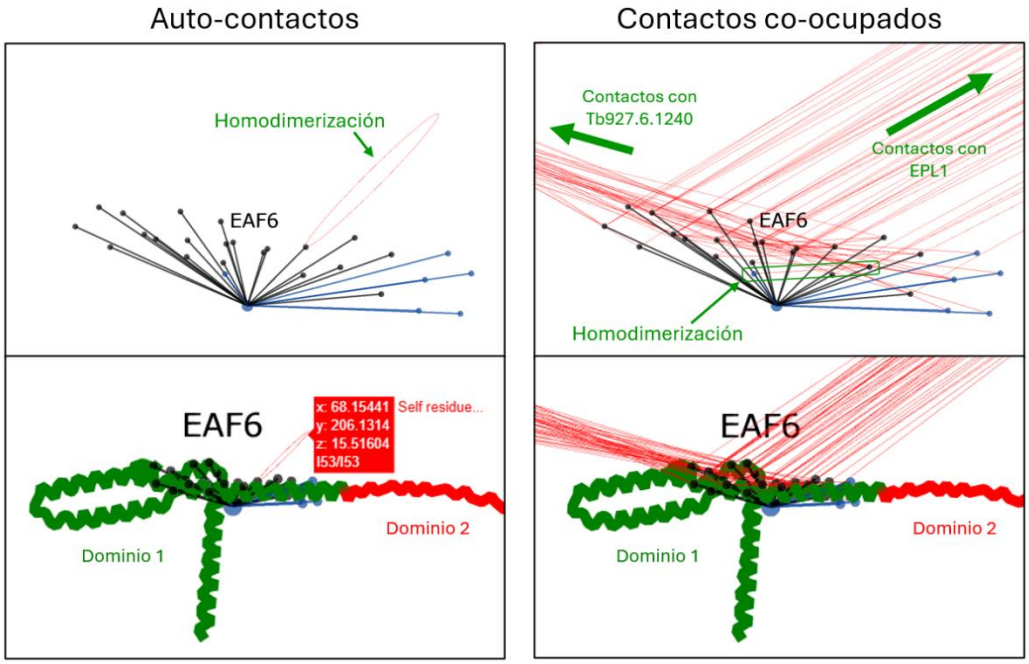

Fig. 31: Contactos de EAF6 con otras proteínas en el grafo 3D. EAF6 interactúa consigo misma, con Tb927.6.1240 y con EPL1, todo a través de la misma superficie de interacción sobre el dominio 1 (residuos de contacto negros).

Si bien MM cuenta con una amplia variedad de funciones que permiten convertir los datos de AF2m en objetos computacionales de Python con cuales interactuar, las funciones básicas del paquete hasta aquí descritas permiten inferir con cierto grado de certeza los coeficientes estequiométricos más probables para el conjunto de estructuras del *dataset*. En nuestro ejemplo, es posible deducir que el complejo consiste en un complejo heterooligomérico donde todos los componentes poseen un coeficiente estequiométrico unitario, ya que no existen interacciones de homooligomerización que persistan en los N-meros. Además, al combinar estas observaciones con lo que se conoce de la bibliografía del complejo NuA4 en otros organismos (Devoucoux, 2022; Wang, 2018), sabemos que este complejo forma 3 subcomplejos con subunidades de estequiometrías unitarias que son capturados perfectamente por las interacciones estáticas del grafo de PPIs (**Fig. 32**), lo cual sería difícil de interpretar si únicamente contáramos con el conjunto de interacciones que superan las métricas sin incorporar las interpretaciones dinámicas (similar a la **Fig. 25**).

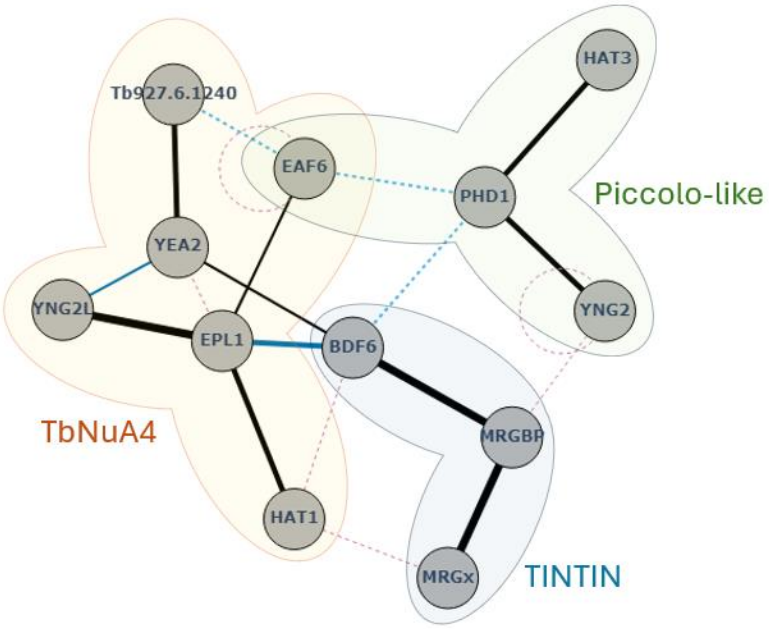

Fig. 32: Subcomplejos NuA4 de *T. brucei*.

Así, una vez deducido la estequiometría, MM cuenta con la función `generate_filesystem_for_CombFold` que permite, como su nombre lo indica, generar el sistema de archivos necesarios para ejecutar CombFold. La función requiere como input un archivo tabular que tenga los coeficientes estequiométricos de cada dominio de cada proteína (Q values), la localización de los PDB con los 2-meros/N-meros, un nombre para crear un directorio de salida y el objeto `sliced_PAE_and_pLDDTs` generado previamente:

```
# Para el subcomplejo NuA4 + TINTIN
mm.generate_filesystem_for_CombFold(xlsx_Qvalues = "NuA4_combination.xlsx",
                                    out_folder = "CombFold_NuA4",
                                    sliced_PAE_and_pLDDTs = sliced_PAE_and_pLDDTs ,
                                    AF2_2mers = AF2_2mers,
                                    AF2_Nmers = AF2_Nmers)
```

El sistema de archivos contendrá todo lo necesario para ejecutar CombFold según lo indicado en su página de GitHub (https://github.com/dina-lab3D/CombFold). El resultado del ensamblado combinatorio donde se agregó el cofactor de las proteínas histona acetiltransferasas (HAT) CMC junto a algunas anotaciones sobre los dominios identificados se muestra en la **Fig. 33**.

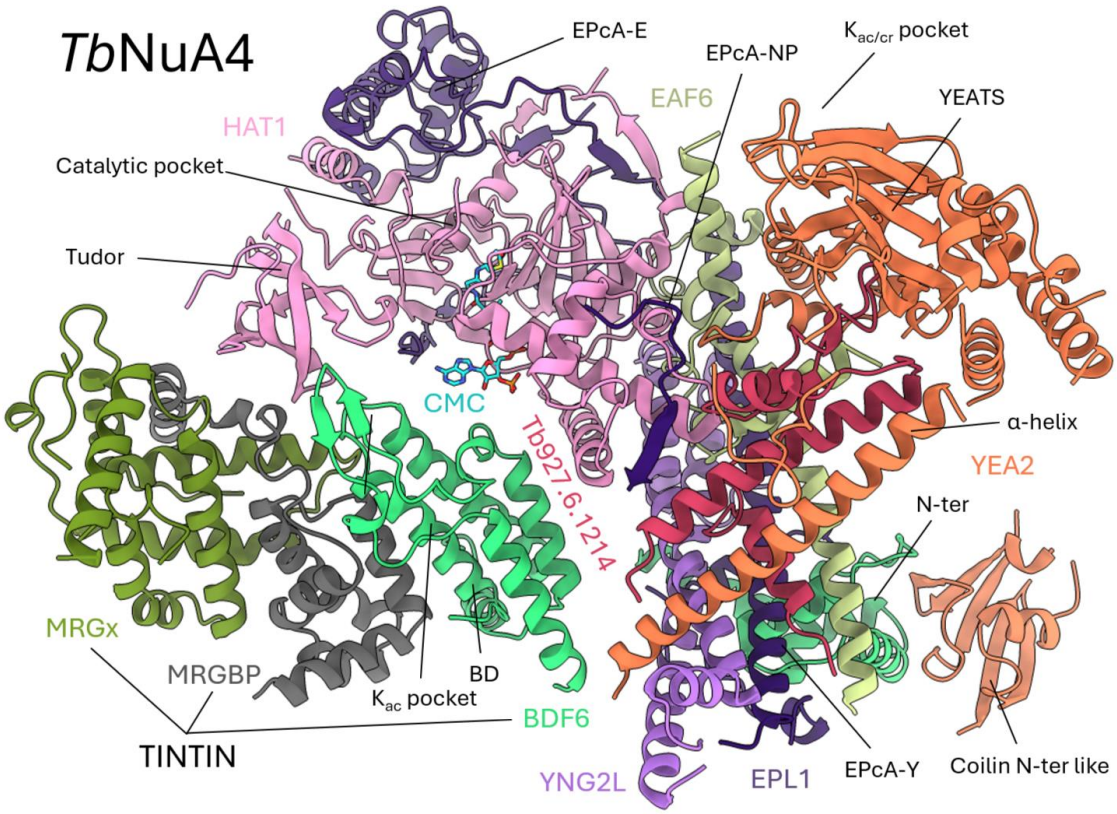

Fig. 33: Ensamblado combinatorio de NuA4.

6. Conclusiones

Entre las conclusiones de este estudio se destacan varios logros significativos derivados del desarrollo y validación de MultimerMapper en el contexto de la bioinformática estructural de tripanosomátidos. Por ejemplo, se logró crear un dataset robusto que fue crucial para entrenar un clasificador competitivo que identifica interacciones proteína-proteína, utilizando predicciones generadas por AF2m. Este logro no solo facilitó la construcción de un marco computacional efectivo para extraer y analizar información relevante de las predicciones de AF2m, sino que también permitió aplicar con éxito la lógica planteada en la hipótesis de las interacciones dinámicas e incorporarla dentro de un entorno computacional.

Además, se logró integrar distintos módulos que transforman esta información en representaciones visuales tanto en 2D (a nivel de PPIs) como en representaciones 3D (a nivel de contactos residuo-residuo), a pesar de que estas últimas aún están en proceso de integrar los contactos de los N-meros y su comportamiento dinámico.

Aunque se documentaron extensivamente muchas de las funciones y clases de MultimerMapper, se identificaron áreas que requieren refinamiento antes de su distribución como paquete en PyPI. Actualmente, MultimerMapper puede instalarse utilizando Anaconda 3 y es compatible con cualquier sistema operativo capaz de ejecutar Python 3.11. Además, para los usuarios que solo estén interesados en generar las visualizaciones 2D y 3D o que no posean conocimientos de Python, se integró todo el pipeline en un único programa ejecutable desde la línea de comandos. El mensaje de ayuda de MultimerMapper *command-line* se muestra en la **Fig. 34**.

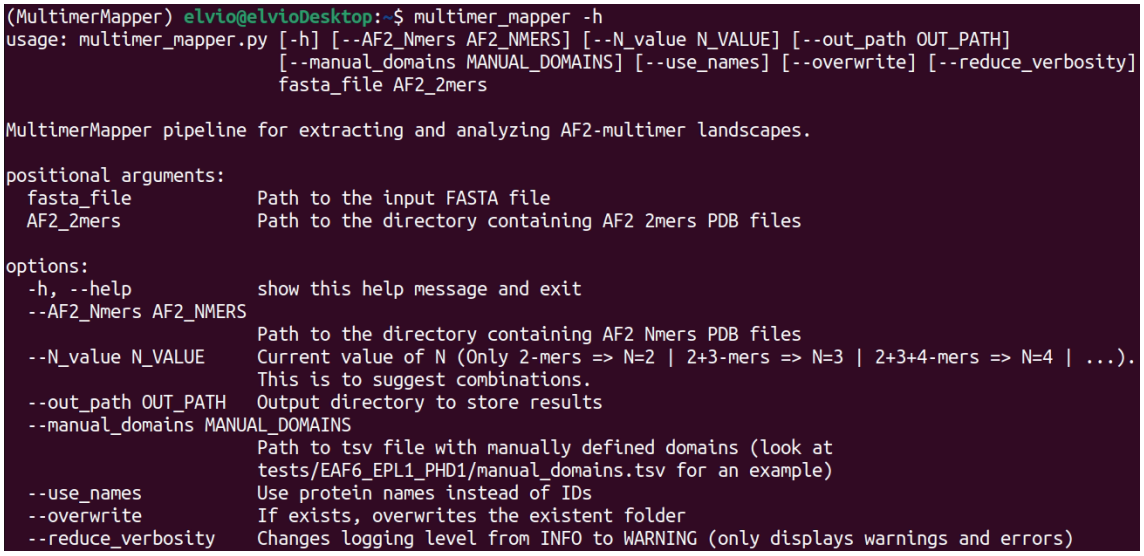

Fig. 34: Mensaje de ayuda de MultimerMapper command-line.

La utilidad de esta herramienta para extraer información de datos bioinformáticos estructurales es innegable, especialmente considerando la falta de alternativas disponibles en el campo hasta la fecha. MultimerMapper tiene el potencial de convertirse en una herramienta indispensable en la investigación bioinformática estructural, ofreciendo nuevas perspectivas sobre la dinámica y ensamblaje de grandes complejos proteicos. La exploración de interacciones dinámicas podría abrir la puerta a nuevos algoritmos y metodologías que revelen caminos de ensamblaje y estequiometrías que de otro modo podrían pasar desapercibidos en los análisis estáticos tradicionales.

Este trabajo sienta las bases para futuras investigaciones y desarrollos en el campo de la bioinformática estructural, promoviendo el uso de MultimerMapper como una herramienta fundamental para investigaciones futuras sobre complejos proteicos dinámicos. Además de encontrarse en activo desarrollo, MultimerMapper está siendo

empleado activamente para investigar el comportamiento dinámico de diversos complejos de tripanosomátidos, identificando proteínas y sus superficies de interacción como posibles blancos terapéuticos. Su integración en estas investigaciones es crucial para acelerar el descubrimiento y desarrollo de fármacos capaces de interferir con el funcionamiento normal de estos complejos y tratar las enfermedades causadas por estos parásitos.

7. Bibliografía

Basu, S., & Wallner, B. (2016). DockQ: A quality measure for protein-protein docking models. *PLoS ONE*, *11*(8), 1–9. https://doi.org/10.1371/journal.pone.0161879

Bryant, P., Pozzati, G., & Elofsson, A. (2022). Improved prediction of protein-protein interactions using AlphaFold2. *Nature Communications*, *13*(1), 1–11. https://doi.org/10.1038/s41467-022-28865-w

Devoucoux, M., Roques, C., Lachance, C., Lashgari, A., Joly-Beauparlant, C., Jacquet, K., Alerasool, N., Prudente, A., Taipale, M., Droit, A., Lambert, J.-P., Hussein, S. M. I., & Côté, J. (2022). MRG proteins are shared by multiple protein complexes with distinct functions. *Molecular & Cellular Proteomics*, *21*, 100253. https://doi.org/10.1016/j.mcpro.2022.100253

Evans, R., O'Neill1, M., Pritzel1, A., Antropova, N., Senior, A., Green, T., Žídek, A., Bates, R., Blackwell, S., Yim, J., Ronneberger, O., Bodenstein, S., Zielinski, M., Bridgland, A., Potapenk, A., Clancy, E., Kohli, P., Jumper, J., & Hassabis, D. (2021). Protein complex prediction with AlphaFold2-Multimer. *Methods in Molecular Biology*, *804*, 297–312. https://doi.org/doi.org/10.1101/2021.10.04.463034

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., … Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, *596*(7873), 583–589. https://doi.org/10.1038/s41586-021-03819-2

Mirdita, M., Schütze, K., Moriwaki, Y., Heo, L., Ovchinnikov, S., & Steinegger, M. (2022). ColabFold: making protein folding accessible to all. *Nature Methods*, *19*(6), 679–682. https://doi.org/10.1038/s41592-022-01488-1

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*(3), 443–453. https://doi.org/https://doi.org/10.1016/0022-2836(70)90057-4

Shor, B., & Schneidman-Duhovny, D. (2024). CombFold: predicting structures of large protein assemblies using a combinatorial assembly algorithm and

AlphaFold2. *Nature Methods*, *21*(March). https://doi.org/10.1038/s41592-024-02174-0

Staneva, D. P., Carloni, R., Auchynnikava, T., Tong, P., Rappsilber, J., Jeyaprakash, A. A., Matthews, K. R., & Allshire, R. C. (2021). A systematic analysis of Trypanosoma brucei chromatin factors identifies novel protein interaction networks associated with sites of transcription initiation and termination . *Genome Research*, *31*(11), 2138–2154. https://doi.org/10.1101/gr.275368.121

Wang, X., Ahmad, S., Zhang, Z., Côté, J., & Cai, G. (2018). Architecture of the Saccharomyces cerevisiae NuA4/TIP60 complex. *Nature Communications*, *9*(1), 1–11. https://doi.org/10.1038/s41467-018-03504-5

Wheeler, R. J. (2021). A resource for improved predictions of Trypanosoma and Leishmania protein three-dimensional structure. *PLoS ONE*, *16*(11 November), 1–12. https://doi.org/10.1371/journal.pone.0259871

Xie, T., Zmyslowski, A. M., Zhang, Y., & Radhakrishnan, I. (2015). Structural Basis for Multi-specificity of MRG Domains. *Structure*, *23*(6), 1049–1057. https://doi.org/10.1016/j.str.2015.03.020

Zhou, T.-M., Wang, S., & Xu, J. (2018). Deep learning reveals many more inter-protein residue-residue contacts than direct coupling analysis. *BioRxiv*, 240754. https://api.semanticscholar.org/CorpusID:89793015

8. Anexo

Tabla Suplementaria 1: Valores de corte para diferentes valores de FPR y diferentes valores de modelos usados como *cutoffs*.

FPR_cutoff	dual_cutoff	N_models	cutoff1	PAE_cutoff	max_sensitiviy
0.01	ipTM + PAE	1	0.670	2.22	37.9
0.01	pDockQ + PAE	1	0.101	1.61	36.1
0.01	ipTM + PAE	2	0.290	3.15	50.3
0.01	pDockQ + PAE	2	0.070	3.15	50.3
0.01	ipTM + PAE	3	0.280	3.82	49.1
0.01	pDockQ + PAE	3	0.051	3.82	49.1
0.01	ipTM + PAE	4	0.280	6.05	51.5
0.01	pDockQ + PAE	4	0.050	6.05	51.5
0.01	ipTM + PAE	5	0.260	8.26	49.1
0.01	pDockQ + PAE	5	0.051	8.26	49.1
0.02	ipTM + PAE	1	0.310	2.22	48.5
0.02	pDockQ + PAE	1	0.109	2.43	49.1
0.02	ipTM + PAE	2	0.290	3.76	52.1
0.02	pDockQ + PAE	2	0.050	3.76	52.1
0.02	ipTM + PAE	3	0.240	6.16	54.4
0.02	pDockQ + PAE	3	0.051	6.16	54.4
0.02	ipTM + PAE	4	0.210	7.56	53.3
0.02	pDockQ + PAE	4	0.050	7.56	53.3
0.02	ipTM + PAE	5	0.200	9.59	50.9
0.02	pDockQ + PAE	5	0.051	9.59	50.9
0.05	ipTM + PAE	1	0.310	2.70	51.5
0.05	pDockQ + PAE	1	0.070	2.97	53.3
0.05	ipTM + PAE	2	0.280	6.88	57.4
0.05	pDockQ + PAE	2	0.037	6.88	57.4
0.05	ipTM + PAE	3	0.240	8.99	57.4
0.05	pDockQ + PAE	3	0.022	8.99	57.4
0.05	ipTM + PAE	4	0.160	11.40	56.2
0.05	pDockQ + PAE	4	0.050	12.23	57.4
0.05	ipTM + PAE	5	0.140	12.97	57.4
0.05	pDockQ + PAE	5	0.028	12.97	57.4
0.10	ipTM + PAE	1	0.280	5.61	60.4
0.10	pDockQ + PAE	1	0.109	7.16	61.5
0.10	ipTM + PAE	2	0.160	9.60	62.1
0.10	pDockQ + PAE	2	0.070	10.46	63.9
0.10	ipTM + PAE	3	0.160	11.54	63.3
0.10	pDockQ + PAE	3	0.022	11.51	63.3
0.10	ipTM + PAE	4	0.150	12.94	60.9
0.10	pDockQ + PAE	4	0.030	12.94	60.9
0.10	ipTM + PAE	5	0.130	16.39	60.9
0.10	pDockQ + PAE	5	0.028	15.22	60.4