



MAESTRÍA EN INTELIGENCIA ARTIFICIAL APLICADA

Materia:

Proyecto Integrador

Avance 6

EQUIPO 29

Miguel Angel Favela Corella A00818504

Kevin Balderas Sánchez A01795149

José Manuel García Ogarrio A01795147

Sponsor del Proyecto

- **Dr. Juan Arturo Nolazco** – Director del Data Science Hub, Tecnológico de Monterrey.

Colaborador Invitado

- **Dr. Carlos Alberto Brizuela Rodríguez** – Investigador, CICESE (Centro de Investigación)

Contexto del proyecto

El presente proyecto tiene como objetivo aplicar herramientas de inteligencia artificial y biología computacional para el diseño y análisis de proteínas con potencial aplicación biomédica. Nuestro foco principal se centra en el estudio del receptor GLP-1R, una proteína de membrana que desempeña un papel esencial en la regulación de la glucosa y del metabolismo energético.

El GLP-1R es activado por el péptido GLP-1, lo que estimula la secreción de insulina y reduce el apetito. Por esta razón, este receptor constituye una diana terapéutica importante en el tratamiento de la diabetes tipo 2 y la obesidad. Comprender su estructura, sus interacciones y posibles moduladores ofrece oportunidades para el diseño racional de nuevas moléculas bioactivas.

Nuestro enfoque combina distintas plataformas computacionales de vanguardia: RFDiffusion, MPNN, AlphaFold, y próximamente Rosetta y REF15, con el propósito de obtener una visión integral del diseño estructural, la viabilidad funcional y las propiedades bioquímicas de las proteínas generadas.

Herramientas

- **RFDiffusion**

RFDiffusion es un modelo generativo basado en el principio de difusión probabilística, diseñado para la creación de nuevas estructuras proteicas de forma controlada y coherente. Este sistema aprende patrones estructurales a partir de proteínas reales y posteriormente puede generar diseños que cumplan con restricciones espaciales o funcionales específicas.

En este proyecto, RFDiffusion se emplea para proponer estructuras candidatas que potencialmente interactúen con el receptor GLP-1R. La finalidad es generar proteínas que puedan actuar como ligandos o moduladores del receptor, conservando estabilidad estructural y compatibilidad con el entorno biológico.

- **ProteinMPNN**

ProteinMPNN sirve para procesar grafos moleculares, donde los nodos representan átomos y las aristas los enlaces químicos. Esta arquitectura permite modelar las interacciones locales y globales entre los componentes de una molécula o proteína.

Su función principal es predecir la secuencia de aminoácidos que mejor se ajusta a una estructura tridimensional predefinida. El modelo recibe como entrada una estructura generada por RFDiffusion y, mediante el intercambio de información entre los nodos del grafo (que representan los residuos de aminoácidos), aprende qué combinaciones son más estables y coherentes con la geometría original.

El resultado son una o varias secuencias candidatas que podrían adoptar la forma estructural deseada. Posteriormente, estas secuencias se validan mediante AlphaFold,

que predice si la secuencia realmente se pliega de manera similar a la estructura original.

- **AlphaFold**

AlphaFold, es una herramienta de predicción estructural que utiliza aprendizaje profundo para determinar la conformación tridimensional de proteínas a partir únicamente de su secuencia de aminoácidos.

Su integración en el proyecto tiene como propósito verificar la estabilidad estructural de las secuencias diseñadas con RFDiffusion y ProteinMPNN. Esta herramienta nos permite visualizar las estructuras resultantes y evaluar si adoptan configuraciones energéticamente estables y coherentes con la función prevista. De esta forma, actúa como una herramienta de validación y análisis estructural.

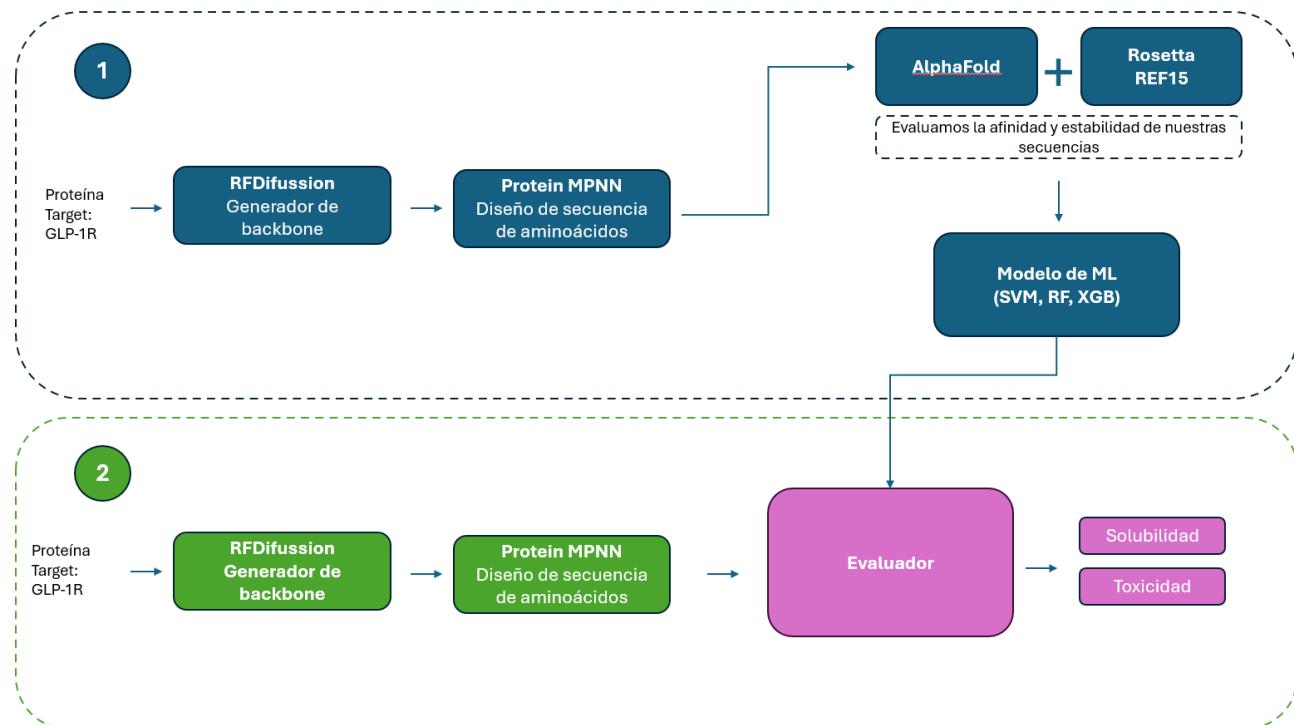
- **Rosetta y REF15**

Rosetta es una de las plataformas computacionales más avanzadas y versátiles para el modelado, diseño y simulación de proteínas. Permite realizar predicciones estructurales, análisis de estabilidad, acoplamiento molecular y optimización energética de biomoléculas. Su funcionamiento se basa en un conjunto de funciones de energía que estiman la estabilidad y la plausibilidad física de una estructura proteica.

Dentro de Rosetta, una de las funciones energéticas más utilizadas es REF15. Esta función integra parámetros físicos y estadísticos que permiten calcular con precisión las energías de interacción, enlaces de hidrógeno, fuerzas hidrofóbicas, interacciones electrostáticas, y energías de solvatación.

En el contexto de este proyecto, Rosetta y REF15 se emplearán conjuntamente para analizar y refinar las proteínas diseñadas mediante RFDiffusion y ProteinMPNN. El objetivo es evaluar no solo la estabilidad estructural y energética, sino también incorporar criterios adicionales relacionados con la toxicidad y la solubilidad de las proteínas generadas.

Etapas y planificación.



Descripción del Flujo de Trabajo

El flujo general del proyecto se divide en dos etapas principales, representadas en el diagrama anterior. Ambas tienen como punto de partida la proteína target GLP-1R, sobre la cual se generan y evalúan nuevas secuencias peptídicas con potencial afinidad estructural y funcional.

Etapa 1: Generación y evaluación inicial de secuencias

En la primera fase del proyecto (azul), se utiliza RFdiffusion como generador del backbone proteico, es decir, la estructura tridimensional base sobre la cual se diseñarán nuevas secuencias. Posteriormente, ProteinMPNN predice en este caso como ejemplo ~1,000/5,000 secuencias de aminoácidos que podrían adoptar dicha estructura.

Estas secuencias se someten a una evaluación estructural y energética mediante AlphaFold y Rosetta (REF15), con el fin de determinar propiedades como la afinidad, estabilidad y compatibilidad estructural.

El principal desafío en esta etapa es el alto costo computacional asociado al cálculo detallado de estas propiedades para miles de secuencias, lo que limita la escalabilidad del proceso.

Para superar esta limitación, se propone entrenar un modelo de ML, empleando algoritmos como SVM, Random Forest o XGBoost, usando un subconjunto representativo de las secuencias generadas.

Este modelo actuará como un predictor de afinidad y estabilidad, reduciendo significativamente el tiempo necesario para evaluar grandes volúmenes de datos sin sacrificar precisión.

Etapa 2: Evaluación masiva mediante modelo predictivo

En la segunda fase (indicada en verde y rosa), se repite el proceso de diseño estructural mediante RFdiffusion y ProteinMPNN, pero esta vez a mayor escala, generando muchas secuencias candidatas.

En lugar de evaluar individualmente cada una mediante simulaciones intensivas, las secuencias se procesan con el modelo evaluador desarrollado en la primera etapa. Este evaluador integra el modelo de ML y permite estimar de manera rápida propiedades clave como la solubilidad y toxicidad.

De esta forma, se priorizan las secuencias con mejores características, reduciendo la carga computacional y enfocando los recursos en aquellas con mayor potencial biotecnológico.

Anexo de Resultados Visuales y Evidencia Computacional

1. Propósito del anexo

Este anexo documenta de manera independiente los resultados visuales obtenidos durante la ejecución del pipeline computacional para el diseño de proteínas orientadas al tratamiento de la diabetes. Se presentan las capturas de ejecución, visualizaciones estructurales y métricas derivadas de los modelos RFdiffusion, ProteinMPNN y AlphaFold, con una descripción técnica clara y contextualizada.

2. Flujo visual de trabajo

2.1 Generación de esqueletos proteicos con RFdiffusion

Imágenes correspondientes a la ejecución local del modelo RFdiffusion, implementado en VSCode mediante notebooks de Jupyter.

Descripción técnica: El modelo generó 10 esqueletos (“backbones”) en formato .pdb a partir de parámetros relacionados con péptidos candidatos para diabetes.

Tiempo de procesamiento: ~35 minutos en GPU GeForce GTX 1650.

Visualizador utilizado: Mol* Viewer.

```

run_rfddiffusion.py

[21] 0.0s
print(result.stderr)

return f"outputs/{output_name}_0.pdb"

[22] 32m 4.4s
[2025-10-05 14:40:49,207][__main__][INFO] - Found GPU with device_name NVIDIA GeForce GTX 1650. Will run RFdiffusion on NVIDIA GeForce GTX 1650
Reading models from /home/kevinwsl/RFdiffusion/rfdiffusion/inference/../../models
[2025-10-05 14:40:49,208][rfdiffusion.inference.model_runners][INFO] - Reading checkpoint from /home/kevinwsl/RFdiffusion/rfdiffusion/inference/../../models
This is inf.conf.ckpt.path
/home/kevinwsl/RFdiffusion/rfdiffusion/inference/../../models/Base_ckpt.pt
Assembling model, diffuser and -preprocess configs from checkpoint
USING MODEL CONFIG: self._conf[model][n_extra_block] = 4
USING MODEL CONFIG: self._conf[model][n_main_block] = 32
USING MODEL CONFIG: self._conf[model][n_ref_block] = 4
USING MODEL CONFIG: self._conf[model][d_msa] = 256
USING MODEL CONFIG: self._conf[model][d_msa_full] = 64
USING MODEL CONFIG: self._conf[model][d_pair] = 128
USING MODEL CONFIG: self._conf[model][d_temp1] = 64
USING MODEL CONFIG: self._conf[model][n_head_msa] = 8
USING MODEL CONFIG: self._conf[model][n_head_pair] = 4
USING MODEL CONFIG: self._conf[model][n_head_temp1] = 4
USING MODEL CONFIG: self._conf[model][n_max_msa] = 256
USING MODEL CONFIG: self._conf[model][n_max_pair] = 128
USING MODEL CONFIG: self._conf[model][n_max_temp1] = 64
USING MODEL CONFIG: self._conf[model][n_max_token] = 256
USING MODEL CONFIG: self._conf[model][n_max_token1] = 64
USING MODEL CONFIG: self._conf[model][n_max_token2] = 64
USING MODEL CONFIG: self._conf[model][n_max_token3] = 64
USING MODEL CONFIG: self._conf[model][n_max_token4] = 64
USING MODEL CONFIG: self._conf[model][n_max_token5] = 64
USING MODEL CONFIG: self._conf[model][n_max_token6] = 64
USING MODEL CONFIG: self._conf[model][n_max_token7] = 64
USING MODEL CONFIG: self._conf[model][n_max_token8] = 64
USING MODEL CONFIG: self._conf[model][n_max_token9] = 64
USING MODEL CONFIG: self._conf[model][n_max_token10] = 64
USING MODEL CONFIG: self._conf[model][n_max_token11] = 64
USING MODEL CONFIG: self._conf[model][n_max_token12] = 64
USING MODEL CONFIG: self._conf[model][n_max_token13] = 64
USING MODEL CONFIG: self._conf[model][n_max_token14] = 64
USING MODEL CONFIG: self._conf[model][n_max_token15] = 64
USING MODEL CONFIG: self._conf[model][n_max_token16] = 64
USING MODEL CONFIG: self._conf[model][n_max_token17] = 64
USING MODEL CONFIG: self._conf[model][n_max_token18] = 64
USING MODEL CONFIG: self._conf[model][n_max_token19] = 64
USING MODEL CONFIG: self._conf[model][n_max_token20] = 64
USING MODEL CONFIG: self._conf[model][n_max_token21] = 64
USING MODEL CONFIG: self._conf[model][n_max_token22] = 64
USING MODEL CONFIG: self._conf[model][n_max_token23] = 64
USING MODEL CONFIG: self._conf[model][n_max_token24] = 64
USING MODEL CONFIG: self._conf[model][n_max_token25] = 64
USING MODEL CONFIG: self._conf[model][n_max_token26] = 64
USING MODEL CONFIG: self._conf[model][n_max_token27] = 64
USING MODEL CONFIG: self._conf[model][n_max_token28] = 64
USING MODEL CONFIG: self._conf[model][n_max_token29] = 64
USING MODEL CONFIG: self._conf[model][n_max_token30] = 64
USING MODEL CONFIG: self._conf[model][n_max_token31] = 64
USING MODEL CONFIG: self._conf[model][n_max_token32] = 64
USING MODEL CONFIG: self._conf[model][n_max_token33] = 64
USING MODEL CONFIG: self._conf[model][n_max_token34] = 64
USING MODEL CONFIG: self._conf[model][n_max_token35] = 64
USING MODEL CONFIG: self._conf[model][n_max_token36] = 64
USING MODEL CONFIG: self._conf[model][n_max_token37] = 64
USING MODEL CONFIG: self._conf[model][n_max_token38] = 64
USING MODEL CONFIG: self._conf[model][n_max_token39] = 64
USING MODEL CONFIG: self._conf[model][n_max_token40] = 64
USING MODEL CONFIG: self._conf[model][n_max_token41] = 64
USING MODEL CONFIG: self._conf[model][n_max_token42] = 64
USING MODEL CONFIG: self._conf[model][n_max_token43] = 64
USING MODEL CONFIG: self._conf[model][n_max_token44] = 64
USING MODEL CONFIG: self._conf[model][n_max_token45] = 64
USING MODEL CONFIG: self._conf[model][n_max_token46] = 64
USING MODEL CONFIG: self._conf[model][n_max_token47] = 64
USING MODEL CONFIG: self._conf[model][n_max_token48] = 64
USING MODEL CONFIG: self._conf[model][n_max_token49] = 64
USING MODEL CONFIG: self._conf[model][n_max_token50] = 64
USING MODEL CONFIG: self._conf[model][n_max_token51] = 64
USING MODEL CONFIG: self._conf[model][n_max_token52] = 64
USING MODEL CONFIG: self._conf[model][n_max_token53] = 64
USING MODEL CONFIG: self._conf[model][n_max_token54] = 64
USING MODEL CONFIG: self._conf[model][n_max_token55] = 64
USING MODEL CONFIG: self._conf[model][n_max_token56] = 64
USING MODEL CONFIG: self._conf[model][n_max_token57] = 64
USING MODEL CONFIG: self._conf[model][n_max_token58] = 64
USING MODEL CONFIG: self._conf[model][n_max_token59] = 64
USING MODEL CONFIG: self._conf[model][n_max_token60] = 64
USING MODEL CONFIG: self._conf[model][n_max_token61] = 64
USING MODEL CONFIG: self._conf[model][n_max_token62] = 64
USING MODEL CONFIG: self._conf[model][n_max_token63] = 64
USING MODEL CONFIG: self._conf[model][n_max_token64] = 64
USING MODEL CONFIG: self._conf[model][n_max_token65] = 64
USING MODEL CONFIG: self._conf[model][n_max_token66] = 64
USING MODEL CONFIG: self._conf[model][n_max_token67] = 64
USING MODEL CONFIG: self._conf[model][n_max_token68] = 64
USING MODEL CONFIG: self._conf[model][n_max_token69] = 64
USING MODEL CONFIG: self._conf[model][n_max_token70] = 64
USING MODEL CONFIG: self._conf[model][n_max_token71] = 64
USING MODEL CONFIG: self._conf[model][n_max_token72] = 64
USING MODEL CONFIG: self._conf[model][n_max_token73] = 64
USING MODEL CONFIG: self._conf[model][n_max_token74] = 64
USING MODEL CONFIG: self._conf[model][n_max_token75] = 64
USING MODEL CONFIG: self._conf[model][n_max_token76] = 64
USING MODEL CONFIG: self._conf[model][n_max_token77] = 64
USING MODEL CONFIG: self._conf[model][n_max_token78] = 64
USING MODEL CONFIG: self._conf[model][n_max_token79] = 64
USING MODEL CONFIG: self._conf[model][n_max_token80] = 64
USING MODEL CONFIG: self._conf[model][n_max_token81] = 64
USING MODEL CONFIG: self._conf[model][n_max_token82] = 64
USING MODEL CONFIG: self._conf[model][n_max_token83] = 64
USING MODEL CONFIG: self._conf[model][n_max_token84] = 64
USING MODEL CONFIG: self._conf[model][n_max_token85] = 64
USING MODEL CONFIG: self._conf[model][n_max_token86] = 64
USING MODEL CONFIG: self._conf[model][n_max_token87] = 64
USING MODEL CONFIG: self._conf[model][n_max_token88] = 64
USING MODEL CONFIG: self._conf[model][n_max_token89] = 64
USING MODEL CONFIG: self._conf[model][n_max_token90] = 64
USING MODEL CONFIG: self._conf[model][n_max_token91] = 64
USING MODEL CONFIG: self._conf[model][n_max_token92] = 64
USING MODEL CONFIG: self._conf[model][n_max_token93] = 64
USING MODEL CONFIG: self._conf[model][n_max_token94] = 64
USING MODEL CONFIG: self._conf[model][n_max_token95] = 64
USING MODEL CONFIG: self._conf[model][n_max_token96] = 64
USING MODEL CONFIG: self._conf[model][n_max_token97] = 64
USING MODEL CONFIG: self._conf[model][n_max_token98] = 64
USING MODEL CONFIG: self._conf[model][n_max_token99] = 64
USING MODEL CONFIG: self._conf[model][n_max_token100] = 64
USING MODEL CONFIG: self._conf[model][n_max_token101] = 64
USING MODEL CONFIG: self._conf[model][n_max_token102] = 64
USING MODEL CONFIG: self._conf[model][n_max_token103] = 64
USING MODEL CONFIG: self._conf[model][n_max_token104] = 64
USING MODEL CONFIG: self._conf[model][n_max_token105] = 64
USING MODEL CONFIG: self._conf[model][n_max_token106] = 64
USING MODEL CONFIG: self._conf[model][n_max_token107] = 64
USING MODEL CONFIG: self._conf[model][n_max_token108] = 64
USING MODEL CONFIG: self._conf[model][n_max_token109] = 64
USING MODEL CONFIG: self._conf[model][n_max_token110] = 64
USING MODEL CONFIG: self._conf[model][n_max_token111] = 64
USING MODEL CONFIG: self._conf[model][n_max_token112] = 64
USING MODEL CONFIG: self._conf[model][n_max_token113] = 64
USING MODEL CONFIG: self._conf[model][n_max_token114] = 64
USING MODEL CONFIG: self._conf[model][n_max_token115] = 64
USING MODEL CONFIG: self._conf[model][n_max_token116] = 64
USING MODEL CONFIG: self._conf[model][n_max_token117] = 64
USING MODEL CONFIG: self._conf[model][n_max_token118] = 64
USING MODEL CONFIG: self._conf[model][n_max_token119] = 64
USING MODEL CONFIG: self._conf[model][n_max_token120] = 64
USING MODEL CONFIG: self._conf[model][n_max_token121] = 64

```

Figura 1. Ejecución del modelo RFdiffusion en entorno local.

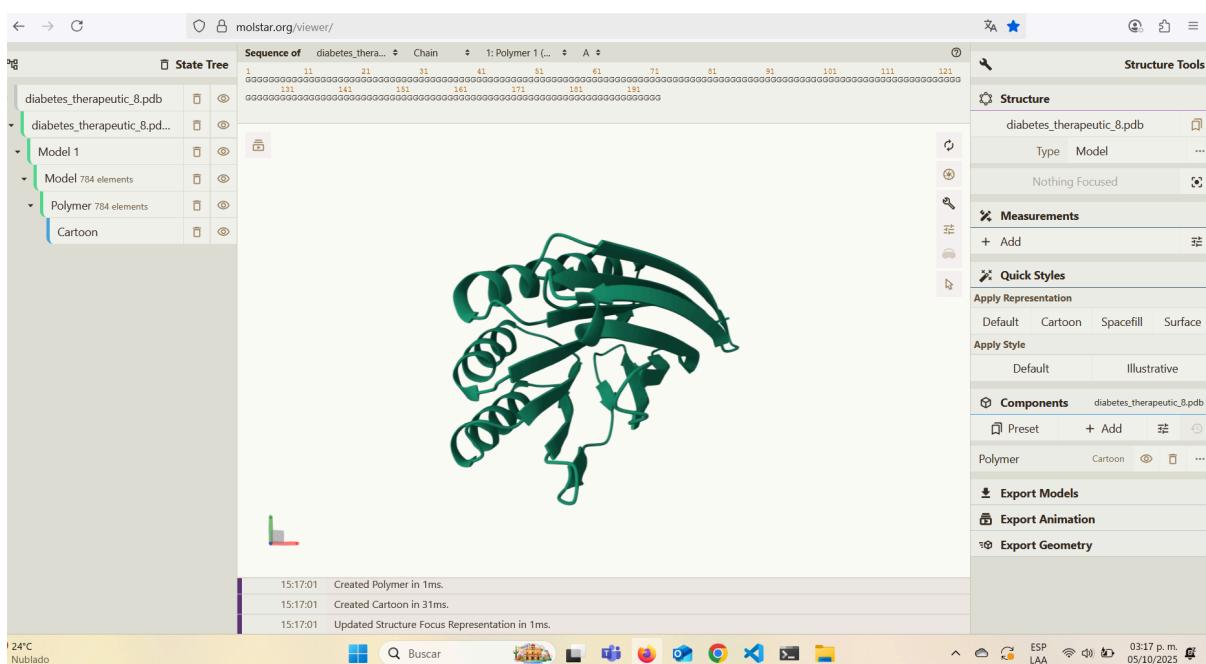


Figura 2. Visualización de estructuras generadas (.pdb) en MolViewer.

2.2 Generación de secuencias aminoacídicas con ProteinMPNN

Descripción técnica: Se utilizó el script `protein_mpnn_run.py` dentro del entorno Conda configurado. A partir de los archivos .pdb generados por RFdiffusion, se obtuvieron secuencias en formato FASTA.

Resultados: Se generaron 10 secuencias con scores entre 1.009 y 1.2303 (valores de log-probabilidad negativa).

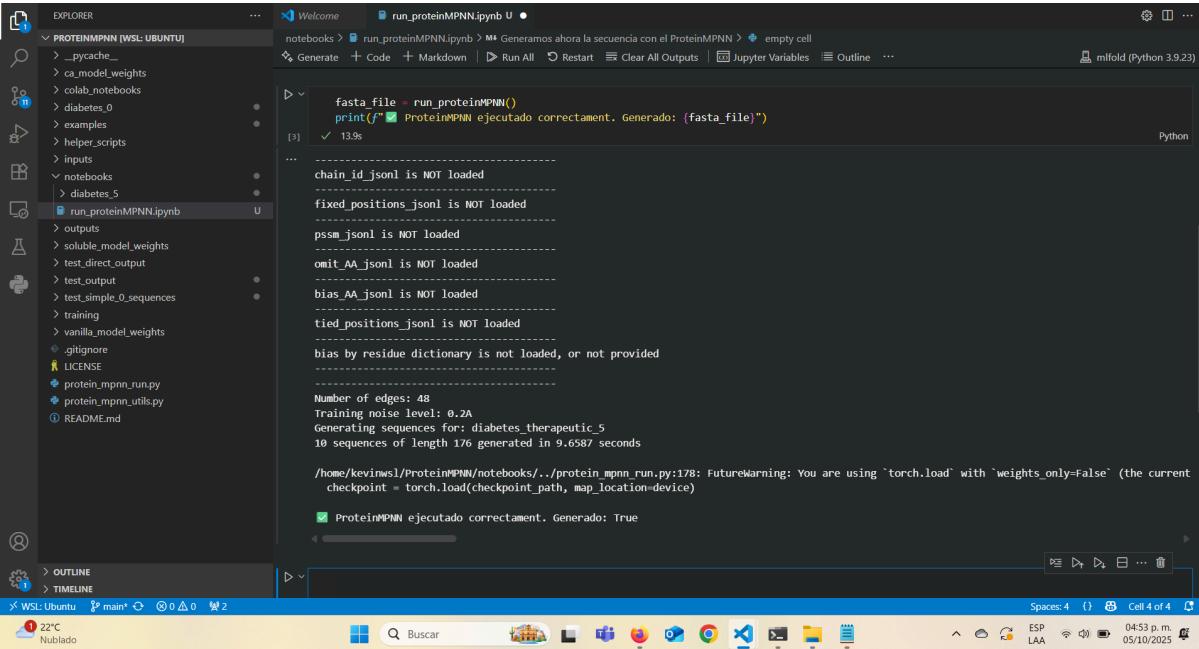


Figura 3. Ejecución de ProteinMPNN para obtención de secuencias.

Figura 4. Visualización de resultados y scores obtenidos.

2.3 Validación estructural con AlphaFold2

Descripción técnica: Se empleó colabfold_batch para validar las secuencias generadas. Para cada muestra, se produjeron cinco modelos estructurales.

Parámetros de rendimiento:

- Total de combinaciones: 55 validaciones.
 - Duración total: ~1h 14min.
 - Métricas utilizadas: pLDDT, pTM y RMSD

Resultados destacados:

- RMSD inicial: 9.16 Å (muestra compleja).
 - RMSD optimizado en monómero simple: 0.31 Å.

```

kevinwsl@LAPGAMKEVIN: ~ + 
2025-10-06 00:37:42,335 alphafold2_ptm_model_3_seed_000 recycle=2 pLDDT=78.6 pTM=0.653 tol=1
2025-10-06 00:38:01,057 alphafold2_ptm_model_3_seed_000 recycle=3 pLDDT=76.8 pTM=0.64 tol=0.278
2025-10-06 00:38:01,057 alphafold2_ptm_model_3_seed_000 tool 76.2s (3 recycles)
2025-10-06 00:38:21,357 alphafold2_ptm_model_4_seed_000 recycle=0 pLDDT=72.2 pTM=0.585
2025-10-06 00:38:40,311 alphafold2_ptm_model_4_seed_000 recycle=1 pLDDT=78.8 pTM=0.669 tol=1.07
2025-10-06 00:38:59,645 alphafold2_ptm_model_4_seed_000 recycle=2 pLDDT=76.9 pTM=0.639 tol=0.427
2025-10-06 00:39:20,699 alphafold2_ptm_model_4_seed_000 recycle=3 pLDDT=77.8 pTM=0.633 tol=0.421
2025-10-06 00:39:20,701 alphafold2_ptm_model_4_seed_000 tool 79.6s (3 recycles)
2025-10-06 00:39:40,135 alphafold2_ptm_model_5_seed_000 recycle=0 pLDDT=82.1 pTM=0.72
2025-10-06 00:40:00,743 alphafold2_ptm_model_5_seed_000 recycle=1 pLDDT=88.9 pTM=0.696 tol=0.287
2025-10-06 00:40:00,867 alphafold2_ptm_model_5_seed_000 recycle=2 pLDDT=78.3 pTM=0.674 tol=0.157
2025-10-06 00:40:39,467 alphafold2_ptm_model_5_seed_000 recycle=3 pLDDT=78.7 pTM=0.674 tol=0.168
2025-10-06 00:40:39,467 alphafold2_ptm_model_5_seed_000 took 78.7s (3 recycles)
2025-10-06 00:40:39,494 reranking models by 'pLDDT' metric
2025-10-06 00:40:39,495 rank_001.alphafold2_ptm_model_1_seed_000 pLDDT=78.9 pTM=0.65
2025-10-06 00:40:39,495 rank_002.alphafold2_ptm_model_5_seed_000 pLDDT=78.7 pTM=0.674
2025-10-06 00:40:39,496 rank_003.alphafold2_ptm_model_2_seed_000 pLDDT=78.0 pTM=0.688
2025-10-06 00:40:39,496 rank_004.alphafold2_ptm_model_4_seed_000 pLDDT=77.8 pTM=0.633
2025-10-06 00:40:39,497 rank_005.alphafold2_ptm_model_3_seed_000 pLDDT=76.8 pTM=0.64
2025-10-06 00:40:40,781 Query 7/11: T_0_i_sample_6_score_1.1905_global_score_1.1905_seq_recovery_0.0057 (length 176)
2025-10-06 00:40:41,341 Sleeping for 5s. Reason: PENDING
2025-10-06 00:40:46,781 Sleeping for 7s. Reason: RUNNING
COMPLETE: 100% | 150/150 [elapsed: 00:13 remaining: 00:00]

```

Figura 5. Ejecución de AlphaFold y generación de modelos.

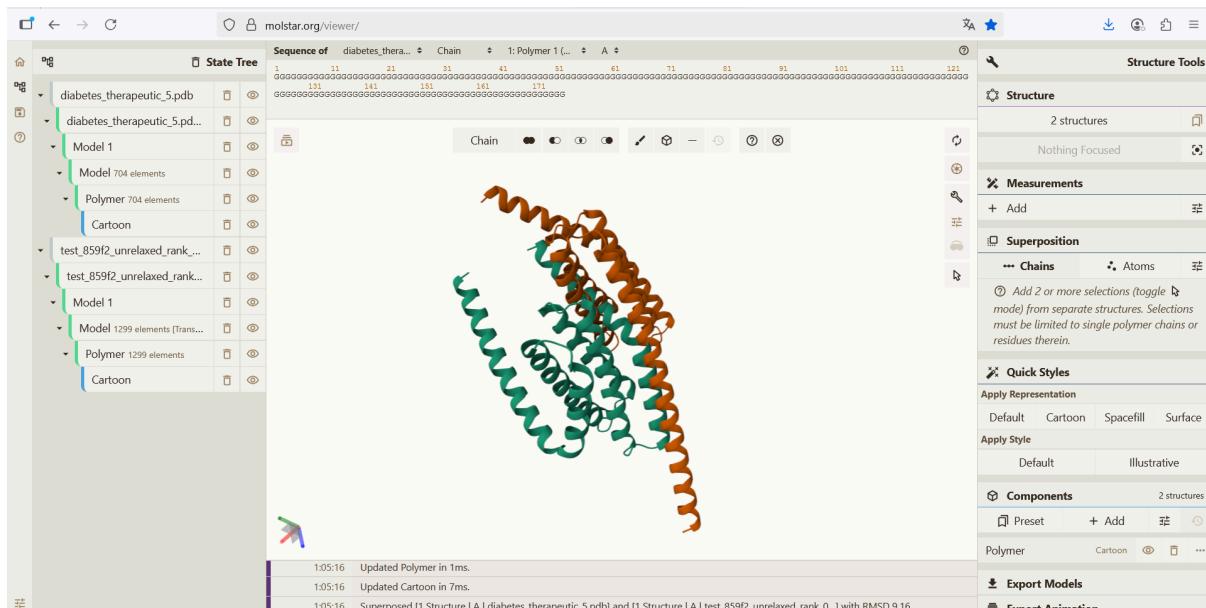


Figura 6. Superposición de estructuras generadas (RFdiffusion vs AlphaFold).

2.4 Integración de pipeline

Flujo de integración logrado:

RFdiffusion → ProteinMPNN → AlphaFold2

Conclusión visual: Se demuestra la viabilidad del pipeline completo para la generación y validación de candidatos peptídicos, dejando como etapa futura la optimización de parámetros y el entrenamiento con Rosetta.

2.5 Obtención energética con Rosetta

Descripción técnica: Se empleó el módulo de PyRosetta nativo de Python para evaluar las nuevas secuencias (archivos pdb) generados previamente por AlphaFold. El script calculate_stability.py crea una función de energía y luego la calcula en base a nuestro .pdb, obteniendo métricas útiles. Pero el segundo script calculate_stability_completed.py aplicá la FastRelax para así obtener métricas importantes como el RMSD que servirán para nuestros modelos futuros evaluadores.

Antes se presenta el pdb utilizado para esta prueba:

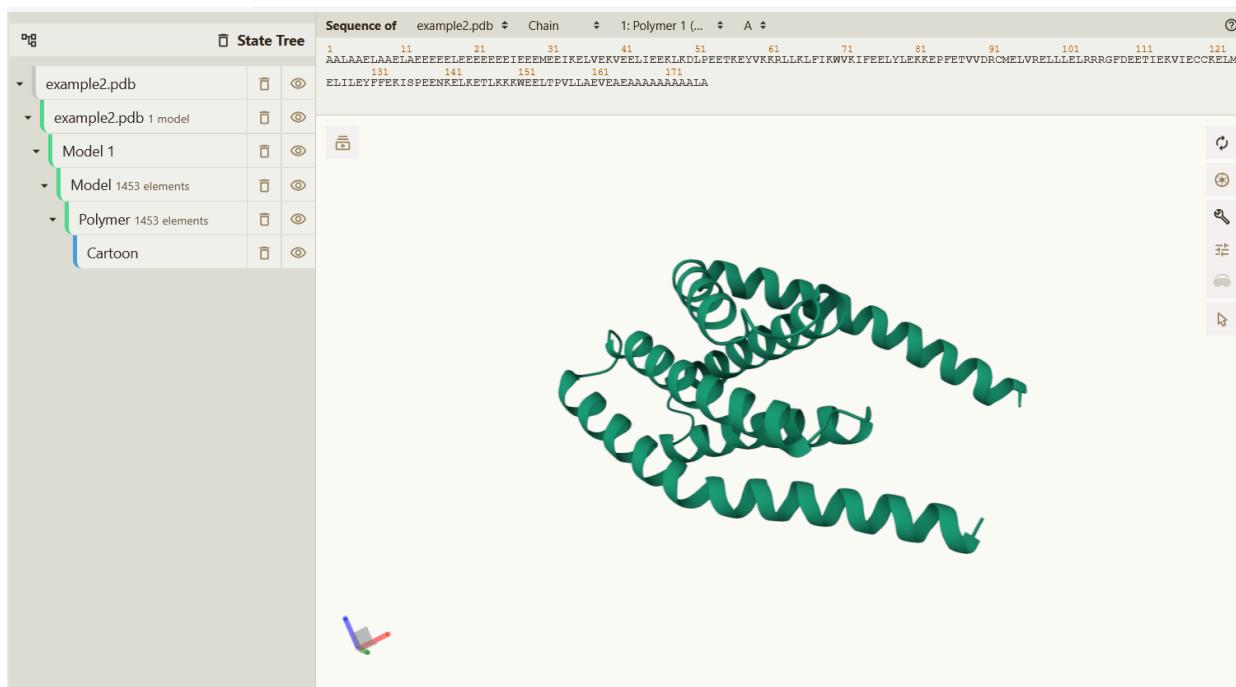


Figura 7. Archivo pdb utilizado para el proceso de Rosetta.

A screenshot of a VS Code terminal window titled "rosetta_scripts_estabilidad [WSL: Ubuntu]". The terminal shows the execution of a Python script named "calculate_stability.py". The script calculates the total energy of a protein structure from a PDB file. The output shows the total energy and a detailed breakdown of energy terms. The terminal also shows the user's command to run the script and the resulting energy value.

Figura 8. Ejecución del script que calcula función de energía full-atom

```

=====
Energía por residuo (top 10 más inestables):
-----
Residuo#  Tipo   Energía (REU)
    65  LYS      174.40
    22  GLU      173.90
    29  GLU      98.55
    72  LYS      96.83
   108  PHE      73.63
    55  THR      70.00
   106  ARG      61.14
    58  TYR      58.10
   132  PHE      40.43
   137  PRO      28.23
=====
Tiempo de ejecución: 0:00:00.697646
Tiempo en segundos: 0.697646
(rosetta env) kevinws1@LAPGAMKEVIN:~/rosetta_scripts_estabilidad/scripts$ 

```

Figura 9. Detalles técnicos de la función de energía full-atom

Parámetros de rendimiento:

- PDB utilizado: Archivo de 171 cadenas
- Duración total: 0.69 segundos
- Métricas utilizadas: Atracción de van der Waals , Solvatación, Electróstatica

Resultados destacados:

- Resumen de métricas a nivel global o general del pdb utilizado
- Análisis de tiempo utilizado durante su ejecución

Ahora se presenta el segundo script creado donde ya usamos Fast-Relax y obtenemos más métricas importantes.

```
(rosetta_env) kevinws1@LAPGAMKEVIN:~/rosetta_scripts_estabilidad/scripts$ python calculate_stability_completed.py
(C) Copyright Rosetta Commons Member Institutions

NOTE: USE OF PyRosetta FOR COMMERCIAL PURPOSES REQUIRES PURCHASE OF A LICENSE
See LICENSE.PyRosetta.md or email license@uw.edu for details

PyRosetta-4 2025 [Rosetta PyRosetta4.Release.python39.ubuntu.cxx11thread.serialization 2025.39+release.c389ea27189ed431c5f30718cf7d86843f
eeab8e 2025-09-24T10:26:10] retrieved from: http://www.pyrosetta.org
Ejecutando FastRelax para optimizar estructura...

=====
RESUMEN DE ESTABILIDAD
=====
Archivo: example2.pdb
Número de residuos: 176

Energía inicial: 951.89 REU
Energía post-relax: -556.77 REU
Cambio energético: -1508.66 REU
Energía/residuo: 5.41 REU/res
RMSD (Ca): 2.22 Å

INTERPRETACIÓN:
-----
⚠️ Estructura MUY INESTABLE (cambio > -5 REU)
⚠️ RMSD alto (2.22 Å) - estructura se deformó mucho
-----

Estructura relajada guardada en: example2.pdb
Tiempo de ejecución: 0:01:11.347443
Tiempo en segundos: 71.347443
(rosetta env) kevinws1@LAPGAMKEVIN:~/rosetta_scripts_estabilidad/scripts$
```

Figura 10. Resultados luego de aplicar Fast-Relax.

```
EXPLORER ... calculate_stability.py 9 calculate_stability_completed.py X
ROSETTA_SCRIPTS_EST... .vscode
scripts > calculate_stability_completed.py > analizar_estructura
10 def analizar_estructura(pdb_file):
    11     if energy_change < 0:
    12         print("✓ Estructura aceptable (cambio < -2 REU)")
    13     else:
    14         print("✓✓ Estructura MUY ESTABLE (energía aumentó o igual)")
    15
    16     if rmsd > 2.0:
    17         print(f"⚠️ RMSD alto ({rmsd:.2f} Å) - estructura se deformó mucho")
    18     elif rmsd > 1.0:
    19         print(f"⚠️ RMSD moderado ({rmsd:.2f} Å)")
    20     else:
    21         print(f"✓ RMSD bajo ({rmsd:.2f} Å) - estructura estable")
    22
    23     print("*" * 60)
    24
    25     # Guardar estructura relajada
    26     output_file = pdb_file.replace('.pdb', '.relaxed.pdb')
    27     relaxed_pose.dump_pdb(output_file)
    28     print(f"\nEstructura relajada guardada en: {output_file}")
    29
    30
    31     return [
    32         'total_energy': total_energy,
```

Figura 11. Estructura de la etapa de Rosetta.

Parámetros de rendimiento:

- PDB utilizado: Archivo de 171 cadenas.
- Duración total: 1.11 minutos.
- Método utilizado: Fast-Relax

Resultados destacados:

- Energía inicial, Energía Post-Relax, Cambio Energético, RMSD
- Análisis de tiempo utilizado durante su ejecución

Por último, en esta etapa se puede concluir que PyRosetta es un módulo muy completo para calcular métricas relacionadas con la energía, así como el método Fast-Relax, el cual es un protocolo que optimiza la geometría molecular. Es decir, luego de los resultados obtenidos mediante AlphaFold, puede existir detalles en su geometría de enlace (no óptima) o energías locales subóptimas. Así, podemos decir que tenemos ya el siguiente flujo logrado:

Flujo de integración logrado:

RFdiffusion → ProteinMPNN → AlphaFold2 → Rosetta

3. Desarrollo de Predictores (AlphaFold y Rosetta)

3.1 Desarrollo de predictor de Alphafold

Para el desarrollo de este modelo predictor primero es necesario identificar entradas y salidas de modelo. Una vez que ya pudimos hacer funcionar los ambientes y el ciclo normal de generación de prospectos para el target que nos interesa, la entrada de alphaFold son las cadenas de secuencias (que serían las salidas luego de correr ProteinMPNN) y de salida tenemos que la variable target es el ipTM (Interface predicted template modelling), la cual es una métrica que nos da alphaFold luego de utilizarlo y que aproximadamente para uno de los primeros ciclos de 100 muestras que generamos, duró cerca de 6.30 horas.

Una vez identificado eso y que justamente queremos hacer un sustituto para encontrar esa métrica, se proponen los siguientes diagramas, que son para la parte de entrenamiento, inferencia y de deployment del predictor.

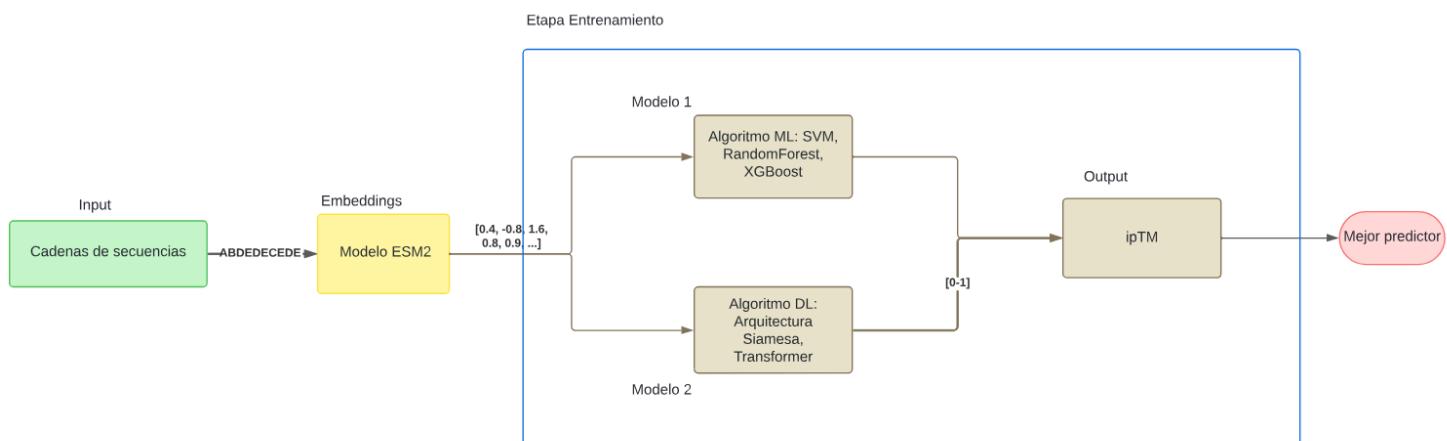


Figura 12. Diagrama de bloques para entrenamiento del predictor de Alphafold

Es importante subrayar que la conversión de secuencias proteicas a representaciones numéricas no puede realizarse mediante técnicas simples como *one-hot encoding*, dado que éstas no capturan las relaciones biológicas ni las dependencias contextuales entre residuos. En su lugar, se propone el uso de modelos de embeddings especializados, capaces de preservar las interacciones latentes y las propiedades bioquímicas implícitas de las cadenas.

Posteriormente, se plantean dos estrategias complementarias de entrenamiento:

- Enfoque clásico de aprendizaje automático, utilizando algoritmos como *Support Vector Machines (SVM)*, *Random Forest* o *XGBoost*, para obtener modelos interpretables y con bajo costo computacional.
- Enfoque de aprendizaje profundo, explorando arquitecturas como *Transformers* o redes *Siamesas*, con el fin de capturar dependencias de largo alcance y relaciones no lineales entre secuencias.

Los resultados de ambos enfoques se evaluarán comparando las predicciones del ipTM frente a los valores obtenidos experimentalmente, utilizando métricas de error como el Root Mean Squared Error (RMSE) para cuantificar el desempeño predictivo.

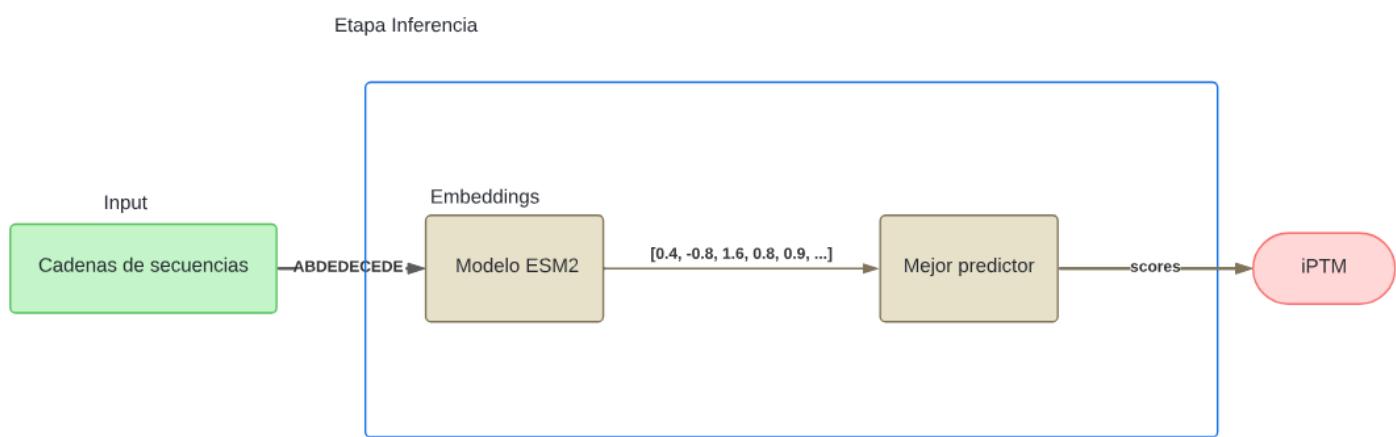


Figura 13. Diagrama de bloques para inferencia del predictor de Alphafold

Finalmente, en la fase de implementación (deployment), se contempla la creación de una API o contenedor modular, capaz de recibir una secuencia de aminoácidos como entrada, ejecutar el proceso de embedding y retornar de forma automatizada la predicción del ipTM. Esta arquitectura permitirá la integración del modelo en flujos de trabajo bioinformáticos más amplios.

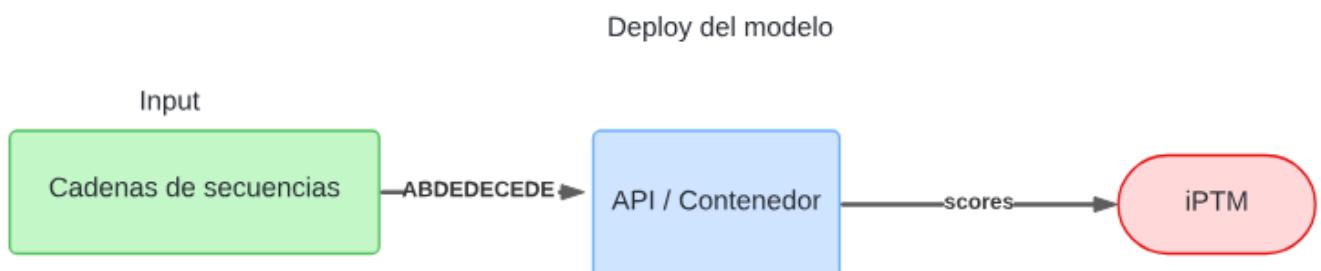


Figura 14. Diagrama de bloques del deploy del predictor de Alphafold

3.2 Desarrollo de organización del deploy del predictor de Alphafold

Ahora, una vez que ya tenemos los prospectos del modelo que sería el predictor de Alphafold, es necesario hacer un ajuste en el repositorio de código que ya tenemos. Esto para facilitar el entendimiento y mejorar la organización de archivos. Como vamos a tener un modelo ganador final, se tiene pensado hacer el deploy mediante un contenedor de Docker. También haciendo notar que uno de los entregables finales será el contenedor que hace todas las etapas para replicación de backbones, es lógico proponer un contenedor para consumir el modelo ganador para esta etapa del predictor de Alphafold. Por último, también una vez con el conocimiento de que el modelo tomará como entrada la cadena y como salida las dos métricas pLDDT e ipTM, es fácil intuir que se proponen dos modos de operación: Uno de tipo cadena y otro de tipo batch. El tipo cadena únicamente tiene como objetivos dar las métricas para esa única cadena que se la da como entrada mediante la consola de comandos, y el tipo batch como su nombre lo indica , cuando tienes un archivo CSV con el formato adecuado, y a la salida tendrás también un CSV pero con las métricas predichas.

```
alphafold_predictor/
|
├── Dockerfile
├── requirements.txt
├── .dockerignore
|
└── src/
    ├── __init__.py
    ├── config.py
    ├── esm_embedder.py
    ├── mlp_predictor.py
    └── batch_predictor.py
|
└── models/
    └── mlp_weights.pth or final_model.joblib
|
└── output/
|
└── run_prediction.py
```

Figura 15. Estructura de archivos para tener el contenedor predictor de Alphafold

Explicando ahora cada archivo y su finalidad tenemos:

- config.py: Define rutas y parámetros centralizados, también variables de entorno como BATCH_SIZE, MAX_SEQ_LENGTH
- requirements.txt: Paquetes o dependencias de python.
- Dockerfile: Es de donde se tomará el build de docker y cuida que para el momento del modelo de embeddings ESM, sólo se descarga en build time y así no cuando se haga run o se hagan duplicidades innecesarias.
- run_prediction.py: De acuerdo al modo de ejecución sea single o batch, hace el proceso correspondiente
- outputs: Guarda los resultados en csv
- models: Tiene el valor de los pesos o parámetros de acuerdo al modelo ganador final.

- mlp_predictor.py: Clase de tipo Arquitectura del Multi Layer Perceptron
- esm_embedder.py: Clase que toma como entrada las cadenas y las pasa por este modelo de embeddings y así obtener los formatos numéricos que necesita el predictor
- batch_predictor.py: Es el archivo que hace la inferencia y únicamente pasa el resultado a run_prediction, para seguir manteniendo modularidad.

Y al final el comando para ejecutar tanto el build como el run en docker serían:

```
docker build -t batch-predictor:latest .
```

Para ver ayuda sobre los comandos y flags disponibles:

```
docker run batch-predictor:latest --help
```

Luego para el modo secuencia para una predicción única:

```
docker run batch-predictor:latest --mode single --sequence "MKTLLAFLFASA"
```

Y para el modo batch:

```
docker run batch-predictor:latest --mode batch --input /app/input.csv
```

3.3 Desarrollo de predictor de Rosetta (Siguiente Semana)

Durante la siguiente etapa, se desarrollará el modelo predictor asociado a Rosetta, el cual tendrá como objetivo estimar métricas energéticas derivadas de las simulaciones *FastRelax*, tales como energía total, RMSD y estabilidad estructural. Este modelo se entrenará utilizando los valores obtenidos experimentalmente por *PyRosetta*, con el propósito de anticipar el comportamiento energético de nuevas secuencias sin necesidad de ejecutar el pipeline completo.

REPORTE DE DISEÑO COMPUTACIONAL DE PROTEÍNAS CON DOCKER (RFdiffusion, ProteinMPNN y Rosetta)

1.- Estructura de carpetas

Crea una carpeta de trabajo principal, por ejemplo:

```
RFDiffusion_Project/
|
├── Dockerfile
├── PyRosetta4.Release.python310.linux.release-387.tar.bz2
├── requirements.txt
|
├── outputs/          # Carpeta donde se guardarán los resultados
├── 1_run_rfddiffusion.py
├── 2_run_mpnn_af.py
└── 3_run_rosetta.py
```

Ingrese al siguiente enlace:

<https://graylab.jhu.edu/download/PyRosetta4/archive/release/PyRosetta4.Release.python310.linux/>

y descargue el archivo: **PyRosetta4.Release.python310.linux.release-387**

2. Creación del contenedor Docker

Construir la imagen

Abre CMD o PowerShell en la carpeta del proyecto y ejecuta:

```
C:\Users\Jose Manuel Garcia\Documents\RFDiffusion>docker build -t rfdiffusion .
```

Esto creará una imagen llamada **rfdiffusion** basada en tu Dockerfile. Incluye dependencias de RFdiffusion, PyRosetta y las librerías necesarias.

```
[+] Building 3.8s (28/28) FINISHED
--> [internal] load build definition from Dockerfile
--> [internal] transferring dockerfile: 3.17kB
--> [internal] resolve image refs for docker.io/nvidia/cuda:12.4.0-devel-ubuntu22.04
--> [internal] load dockerignore
--> [internal] transfer context: 2B
[ 1/23] FROM docker.io/nvidia/cuda:12.4.0-devel-ubuntu22.04@sha256:28326fe73ed47ecf82447e9ca1175db639b931e2348019b2887f35a8d1551656
--> [internal] load build context
--> [internal] transfer files: 779B
--> [ 2/23] RUN apt-get update && apt-get install -y python3 python3-pip git wget aria2 unzip && rm -rf /var/lib/apt/lists/*
--> [ 3/23] RUN pip3 install jedi omegaconf hydra-core icecream pyrsistent pyyaml decorator psutil tqdm numpy scipy networkx requests pillow
--> [ 4/23] WORKDIR /workspace
--> [ 5/23] RUN pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/wheels/cu124
--> [ 6/23] RUN pip3 install git+https://github.com/rdiffusion/rdiffusion@main#egg=rdiffusion
--> [ 7/23] RUN git clone https://github.com/sokrypton/rdiffusion .
--> [ 8/23] RUN pip3 install --no-dependencies dgl-f https://data.dgl.ai/wheels/torch-2.4/cu124/repo.html
--> [ 9/23] RUN pip3 install --no-dependencies einops==0.5.5 opt_einsum_fx
--> [ 10/23] RUN pip3 install git+https://github.com/rdiffusion/rdiffusion@main#egg=rdiffusion
--> [ 11/23] RUN pip3 install git+https://files.ipd.uni.edu/krryptan/ananas && chmod +x ananas
--> [ 12/23] RUN pip3 install git+https://github.com/sokrypton/ColabDesign.git@v1.1
--> [ 13/23] RUN pip3 install --no-dependencies Rdiffusion@main#egg=Rdiffusion
--> [ 14/23] RUN ln -s /usr/local/lib/python3.*/dist-packages/ColabDesign colabdesign
--> [ 15/23] RUN mkdir -p params Rdiffusion/models
--> [ 16/23] RUN aria2c -q -x 16 -d Rdiffusion/models > https://files.ipd.uni.edu/pub/Rdiffusion/f65002a23702ubdd0e176cb93863dc4/Base_ckpt.pt
--> [ 17/23] RUN aria2c -q -x 16 -d Rdiffusion/models http://files.ipd.uni.edu/pub/Rdiffusion/e29311f6f1b1faef907f9ef9fb4ub8328b/Complex_base_ckpt.pt
--> [ 18/23] RUN md5sum -p params && aria2c -q -x 16 https://storage.googleapis.com/alphafold/alphafold_params_2022-12-06.tar && tar -xf alphafold_params_2022-12-06.tar -C params && rm 0.0s
--> [ 19/23] RUN md5sum -p params && aria2c -q -x 16 https://storage.googleapis.com/alphafold/alphafold_params_2022-12-06.tar && tar -xf alphafold_params_2022-12-06.tar -C params && rm 0.0s
--> [ 20/23] COPY PyRosetta4.Release.python310.linux.release-387.tar.b2 /workspace/
--> [ 21/23] RUN cd workspace/PyRosetta4_Release.python310.linux.release-387 && python3 setup.py install
--> [ 22/23] RUN cd workspace/PyRosetta4_Release.python310.linux.release-387 && python3 setup.py install
--> [ 23/23] RUN rm -rf outputs
--> exporting to image
--> exporting layers
--> writing image sha256:b7a57cd319de1619343227af8e067b06c647fcce1354ed99c9d977928c1be5
--> naming to docker.io/library/rfdiffusion
```

3. Configuración del entorno virtual

Aunque Docker ya aísla el entorno, se crea un entorno virtual para posteriormente correr nuestros modelos de machine learning en el proceso predictor.

python3 -m venv protein_env

protein_env\Scripts\activate

Instalar dependencias:

pip install -r requirements.txt

4. Ejecución de los scripts

Cada script de Python se ejecutará dentro del contenedor. La idea es montar las carpetas locales de tu PC en /workspace dentro de Docker

4.1 RFdiffusion — Generación de estructuras

docker run -it --gpus all -v "C:\Users\Jose Manuel

Garcia\Documents\RFDiffusion\outputs":/workspace/outputs -v "C:\Users\Jose Manuel

*Garcia\Documents\RFDiffusion\1_run_rfdiffusion.py":/workspace/1_run_rfdiffusion.py rfdiffusion
python3 1_run_rfdiffusion.py*

Esto:

- Usa tu GPU (--gpus all)
- Monta tu carpeta local de resultados (outputs)
- Ejecuta el script 1_run_rfdiffusion.py dentro del contenedor.

```
(protein_env) C:\Users\Jose Manuel Garcia\Documents\RFDiffusion>docker run -it --gpus all -v "C:\Users\Jose Manuel Garcia\Documents\RFDiffusion\outputs":/workspace/outputs -v "C:\Users\Jose Manuel Garcia\Documents\RFDiffusion\1_run_rfdiffusion.py":/workspace/1_run_rfdiffusion.py rfdiffusion python3 1_run_rfdiffusion.py

=====
== CUDA ==
=====

CUDA Version 12.4.0

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your convenience.

/usr/local/lib/python3.10/dist-packages/torch/cuda/_init_.py:61: FutureWarning: The pynvml package is deprecated. Please install nvidia-ml-py instead. If you did not install pynvml directly, please report this to the maintainers of the package that installed pynvml for you.
import pynvml # type: ignore[import]

=====
VERIFICACIÓN GPU:
CUDA disponible: True
GPU: NVIDIA GeForce RTX 3050 Laptop GPU
Memoria: 4.3 GB

=====
🕒 Descargando 6B3J desde RCSB...
✅ PDB descargado: /workspace/RFDiffusion/6B3J.pdb
🔴 Usando PDB: /workspace/RFDiffusion/6B3J.pdb
Parameters RFDIFFUSION:
Nombre: test_6B3J
Contigs: 12-15/0 R311-337
PDB: /workspace/RFDiffusion/6B3J.pdb
Iteraciones: 200
Hotspot: R312,R313,R314,R315
Diseños: 1
Simetría: None
Orden simetría: None
Cadenas: None
Visual: image
Comando: python3 RFdiffusion/run_inference.py inference.output.prefix=outputs/test_6B3J contigmap.contigs=[12-15/0 R311-337] inference.num_designs=1 diffuser.T=200 inference.dump_pdb=True inference.dump_pdb_path=/dev/shm inference.input_pdb=/workspace/RFDiffusion/6B3J.pdb ppi.hotspot_res=[R312,R313,R314,R315]
=====

Iniciando diseño de proteína...
```

Name	Date modified	Type	Size
test_6B3J_0.trb	10/26/2025 7:17 PM	TRB File	38 KB
test_6B3J_0.pdb	10/26/2025 7:17 PM	Program Debug D...	11 KB
2025-10-27	10/26/2025 9:13 PM	File folder	
traj	10/26/2025 7:17 PM	File folder	

4.2 ProteinMPNN — Generación de secuencias

Luego, usa el modelo 3D generado por RFdiffusion:

```
docker run -it --gpus all -v "C:\Users\Jose Manuel
Garcia\Documents\RFDiffusion\outputs":/workspace/outputs -v "C:\Users\Jose Manuel
Garcia\Documents\RFDiffusion\2_run_mpnn_af.py":/workspace/2_run_mpnn_af.py rfdiffusion python3
2_run_mpnn_af.py
```

```

(protein_env) C:\Users\Jose Manuel Garcia\Documents\RFDiffusion>docker run -it --gpus all -v "C:\Users\Jose Manuel Garcia\Documents\RFDiffusion\outputs":/workspace/outputs -v "C:\Users\Jose Manuel Garcia\Documents\RFDiffusion\2_run_mpnn_af.py":/workspace/2_run_mpnn_af.py rfdiffusion python3 2_run_mpnn_af.py

=====
== CUDA ==
=====

CUDA Version 12.4.0

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your convenience.

=====
● INFORMACION DEL ENTORNO Y ARCHIVOS
=====

■ Directorio actual: /workspace
=====

■ ARCHIVOS EN /workspace/outputs/ (VOLUMEN DOCKER):
    - 2025-10-27/ [carpeta]
        - test_6B3J_0.pdb (10.7 kB)
        - test_6B3J_0.trb (37.7 kB)
        - traj/ [carpeta]
=====

■ PROTEINMPNN + ALPHAFOLD VALIDATION
=====

■ PARÁMETROS:
  PDB: /workspace/outputs/test_6B3J_0.pdb
  Output: /workspace/outputs/test_6B3J
  Contigs: 12-15 R311-337
  Secuencias: 32
  Recycles: 1

=====
[Ejecutando ProteinMPNN + AlphaFold...
[{"pdb": "/workspace/outputs/test_6B3J_0.pdb", "loc": "/workspace/outputs/test_6B3J", "contigs": "12-15R311-337", "copies": 1, "initial_guess": false, "use_multimer": false, "num_recycles": 1, "rm_aa": "C", "num_seqs": 32, "min_aa": "A", "max_aa": "Z", "min_pdb": 1, "max_pdb": 15, "min_sec": 31, "max_sec": 33, "min_rid": 311, "max_rid": 337, "min_cid": 12, "max_cid": 15, "min_rid_c": 311, "max_rid_c": 337, "min_cid_c": 12, "max_cid_c": 15, "min_rid_l": 311, "max_rid_l": 337, "min_cid_l": 12, "max_cid_l": 15, "min_rid_r": 311, "max_rid_r": 337, "min_cid_r": 12, "max_cid_r": 15, "min_rid_t": 311, "max_rid_t": 337, "min_cid_t": 12, "max_cid_t": 15, "min_rid_s": 311, "max_rid_s": 337, "min_cid_s": 12, "max_cid_s": 15, "min_rid_e": 311, "max_rid_e": 337, "min_cid_e": 12, "max_cid_e": 15, "min_rid_f": 311, "max_rid_f": 337, "min_cid_f": 12, "max_cid_f": 15, "min_rid_g": 311, "max_rid_g": 337, "min_cid_g": 12, "max_cid_g": 15, "min_rid_h": 311, "max_rid_h": 337, "min_cid_h": 12, "max_cid_h": 15, "min_rid_i": 311, "max_rid_i": 337, "min_cid_i": 12, "max_cid_i": 15, "min_rid_j": 311, "max_rid_j": 337, "min_cid_j": 12, "max_cid_j": 15, "min_rid_k": 311, "max_rid_k": 337, "min_cid_k": 12, "max_cid_k": 15, "min_rid_l": 311, "max_rid_l": 337, "min_cid_l": 12, "max_cid_l": 15, "min_rid_m": 311, "max_rid_m": 337, "min_cid_m": 12, "max_cid_m": 15, "min_rid_n": 311, "max_rid_n": 337, "min_cid_n": 12, "max_cid_n": 15, "min_rid_o": 311, "max_rid_o": 337, "min_cid_o": 12, "max_cid_o": 15, "min_rid_p": 311, "max_rid_p": 337, "min_cid_p": 12, "max_cid_p": 15, "min_rid_q": 311, "max_rid_q": 337, "min_cid_q": 12, "max_cid_q": 15, "min_rid_r": 311, "max_rid_r": 337, "min_cid_r": 12, "max_cid_r": 15, "min_rid_s": 311, "max_rid_s": 337, "min_cid_s": 12, "max_cid_s": 15, "min_rid_t": 311, "max_rid_t": 337, "min_cid_t": 12, "max_cid_t": 15, "min_rid_u": 311, "max_rid_u": 337, "min_cid_u": 12, "max_cid_u": 15, "min_rid_v": 311, "max_rid_v": 337, "min_cid_v": 12, "max_cid_v": 15, "min_rid_w": 311, "max_rid_w": 337, "min_cid_w": 12, "max_cid_w": 15, "min_rid_x": 311, "max_rid_x": 337, "min_cid_x": 12, "max_cid_x": 15, "min_rid_y": 311, "max_rid_y": 337, "min_cid_y": 12, "max_cid_y": 15, "min_rid_z": 311, "max_rid_z": 337, "min_cid_z": 12, "max_cid_z": 15}, {"design": "1", "mpnn": 1.501, "pldtt": 0.666, "iptm": 0.133, "i_pae": 13.229, "rmsd": 31.484, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLLLLLLLLA", "l": "L"}, {"design": "0:1", "mpnn": 1.684, "pldtt": 0.859, "iptm": 0.843, "i_pae": 1.944, "rmsd": 36.832, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/SELFLTLELLKKKA", "l": "L"}, {"design": "0:2", "mpnn": 1.588, "pldtt": 0.725, "iptm": 0.892, "i_pae": 16.583, "rmsd": 35.828, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLLLLLLLLLKA", "l": "L"}, {"design": "0:3", "mpnn": 1.699, "pldtt": 0.666, "iptm": 0.133, "i_pae": 13.229, "rmsd": 31.484, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLLLLLLLLLKA", "l": "L"}, {"design": "0:4", "mpnn": 1.699, "pldtt": 0.750, "iptm": 0.897, "i_pae": 16.324, "rmsd": 33.675, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/EELLRLLELLKKKA", "l": "L"}, {"design": "0:5", "mpnn": 1.594, "pldtt": 0.793, "iptm": 0.893, "i_pae": 15.831, "rmsd": 35.19, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/EELLKLLELLKKKA", "l": "L"}, {"design": "0:6", "mpnn": 1.587, "pldtt": 0.651, "iptm": 0.102, "i_pae": 14.706, "rmsd": 33.287, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLTTLLLLLLAA", "l": "L"}, {"design": "0:7", "mpnn": 1.537, "pldtt": 0.611, "iptm": 0.121, "i_pae": 13.943, "rmsd": 33.908, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLTTLLLLLLAA", "l": "L"}, {"design": "0:8", "mpnn": 1.594, "pldtt": 0.618, "iptm": 0.099, "i_pae": 15.893, "rmsd": 31.689, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/SLLVVLLLLLLLAA", "l": "L"}, {"design": "0:9", "mpnn": 1.536, "pldtt": 0.666, "iptm": 0.133, "i_pae": 13.229, "rmsd": 31.484, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLLLLLLLLLAA", "l": "L"}, {"design": "0:10", "mpnn": 1.501, "pldtt": 0.710, "iptm": 0.899, "i_pae": 0.871, "rmsd": 17.854, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/SSLLEKLELLKKKA", "l": "L"}, {"design": "0:11", "mpnn": 1.684, "pldtt": 0.798, "iptm": 0.169, "i_pae": 12.854, "rmsd": 29.377, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LELLLTTLLLLLLA", "l": "L"}, {"design": "0:12", "mpnn": 1.611, "pldtt": 0.674, "iptm": 0.101, "i_pae": 15.551, "rmsd": 33.559, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/GLLLTTLLLLLLA", "l": "L"}, {"design": "0:13", "mpnn": 1.681, "pldtt": 0.789, "iptm": 0.108, "i_pae": 14.489, "rmsd": 38.301, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/GLLLTTLLLLLLA", "l": "L"}, {"design": "0:14", "mpnn": 1.545, "pldtt": 0.666, "iptm": 0.133, "i_pae": 14.119, "rmsd": 38.071, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/GLLLTTLLLLLLA", "l": "L"}, {"design": "0:15", "mpnn": 1.522, "pldtt": 0.666, "iptm": 0.133, "i_pae": 13.229, "rmsd": 31.184, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLTTLLLLLLA", "l": "L"}, {"design": "0:16", "mpnn": 1.511, "pldtt": 0.672, "iptm": 0.108, "i_pae": 13.352, "rmsd": 26.176, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/SLLLTTLLLLLLA", "l": "L"}, {"design": "0:17", "mpnn": 1.656, "pldtt": 0.599, "iptm": 0.111, "i_pae": 15.418, "rmsd": 31.339, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/SLLLTTLLLLLLA", "l": "L"}, {"design": "0:18", "mpnn": 1.628, "pldtt": 0.697, "iptm": 0.182, "i_pae": 15.168, "rmsd": 31.262, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/GLLLTTLLLLLLA", "l": "L"}, {"design": "0:19", "mpnn": 1.591, "pldtt": 0.659, "iptm": 0.138, "i_pae": 13.004, "rmsd": 33.258, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLTTLLLLLLA", "l": "L"}, {"design": "0:20", "mpnn": 1.632, "pldtt": 0.773, "iptm": 0.088, "i_pae": 16.886, "rmsd": 39.189, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/SLELLLELLKKKA", "l": "L"}, {"design": "0:21", "mpnn": 1.597, "pldtt": 0.666, "iptm": 0.133, "i_pae": 13.229, "rmsd": 31.484, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLTTLLLLLLA", "l": "L"}, {"design": "0:22", "mpnn": 1.558, "pldtt": 0.697, "iptm": 0.106, "i_pae": 15.048, "rmsd": 30.557, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/GLLLTTLLLLLLA", "l": "L"}, {"design": "0:23", "mpnn": 1.588, "pldtt": 0.688, "iptm": 0.124, "i_pae": 13.714, "rmsd": 31.567, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLTTLLLLLLA", "l": "L"}, {"design": "0:24", "mpnn": 1.526, "pldtt": 0.659, "iptm": 0.138, "i_pae": 13.049, "rmsd": 33.258, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/SLLLTTLLLLLLA", "l": "L"}, {"design": "0:25", "mpnn": 1.678, "pldtt": 0.672, "iptm": 0.121, "i_pae": 14.521, "rmsd": 30.486, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/SLLLTTLLLLLLA", "l": "L"}, {"design": "0:26", "mpnn": 1.704, "pldtt": 0.788, "iptm": 0.071, "i_pae": 17.779, "rmsd": 41.076, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/SELLKKLLETKKKA", "l": "L"}, {"design": "0:27", "mpnn": 1.609, "pldtt": 0.713, "iptm": 0.108, "i_pae": 14.694, "rmsd": 34.528, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/GLLLTTLLLLLLA", "l": "L"}, {"design": "0:28", "mpnn": 1.516, "pldtt": 0.666, "iptm": 0.133, "i_pae": 13.229, "rmsd": 31.484, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLTTLLLLLLA", "l": "L"}, {"design": "0:29", "mpnn": 1.596, "pldtt": 0.685, "iptm": 0.122, "i_pae": 13.922, "rmsd": 33.482, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/LLLLTTLLLLLLA", "l": "L"}, {"design": "0:30", "mpnn": 1.665, "pldtt": 0.807, "iptm": 0.084, "i_pae": 18.175, "rmsd": 33.977, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/FELLKLLELLKKKA", "l": "L"}, {"design": "0:31", "mpnn": 1.604, "pldtt": 0.627, "iptm": 0.112, "i_pae": 15.112, "rmsd": 34.332, "lp": "LPILFAIGVNFLIFPVRCVTVSPLKA/GLLLTTLLLLLLA", "l": "L"}]

● Validación en paralelo en 190.5s
=====

■ ARCHIVOS GENERADOS:
  ✓ mpnn_results.csv (4.6 kB)
  ✓ best.pdb (25.1 kB)
  ✓ best_design.pdb (25.0 kB)
  ✓ design.fasta (3.7 kB)
=====

TIEMPO TOTAL DE EJECUCIÓN: 190.56 segundos

```

ProteinMPNN: genera secuencias de aminoácidos que se ajustan a la estructura 3D dada.

AlphaFold: predice cómo se doblaría esa secuencia, y sirve para verificar si la secuencia propuesta realmente adopta la forma deseada.

4.3 Rosetta — Refinamiento estructural

Finalmente, ejecuta el tercer script dentro del mismo contenedor:

```
docker run -it --gpus all -v "C:\Users\Jose Manuel
```

Garcia\Documents\RFDifussion\outputs":\workspace

Garcia\Documents\RFDifussion\3_run_rosetta.py":/workspace/3_run_rosetta.py rfdiffusion python3 3_run_rosetta.py

```
(protein_env) C:\Users\Jose Manuel Garcia\Documents\RFDiffusion>docker run -it --gpus all -v "C:\Users\Jose Manuel Garcia\Documents\RFDiffusion\outputs":/workspace/outputs -v "C:\Users\Jose Manuel Garcia\Documents\RFDiffusion\3_run_rosetta.py":/workspace/outputs -v "C:\Users\Jose Manuel Garcia\Documents\RFDiffusion\outputs":/workspace/outputs
=====
== CUDA ==
=====

CUDA Version 12.4.0

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your convenience.

PyRosetta-4
Created in JHU by Sergey Lyskov and PyRosetta Team
(C) Copyright Rosetta Commons Member Institutions

NOTE: USE OF PyRosetta FOR COMMERCIAL PURPOSES REQUIRE PURCHASE OF A LICENSE
See LICENSE.PyRosetta.md or email license@uw.edu for details

PyRosetta-4 2024 [Rosetta PyRosetta4_Release.python310.linux_2024.39+release.59628fb5bc09f1221e1642f1f8d157ce40b1410 2024-09-23T07:49:48] retrieved from: http://www.pyrosetta.org
● INICIANDO ANÁLISIS ROSETTA COMPLETO
=====
Encontrados 32 archivos PDB
=====
# ANÁLISIS ROSETTA COMPLETO: design0_n0.pdb
=====
■ INFORMACIÓN BÁSICA:
Residuos: 41
Energía REF15: 199.65 REU
Energía/residuo: 4.65 REU/res

● DESGLOSE ENERGÉTICO:
fa_attr : -177.98 REU
fa_rep : 159.15 REU
fa_sol : 131.69 REU
fa_elec : -40.28 REU
hbond_sc : 8.66 REU
hbond_bb_sc : 0.00 REU
utriga : 9.46 REU
rama_prep : -4.99 REU
p_aa_pp : -6.12 REU
```

🏆 REPORTE COMPARATIVO – MEJORES DISEÑOS					
Archivo	Residues	Energy/Res	RMSD	Estado	
1. design0_n10.pdb	41	3.04	8.74	PROBLEMA	
2. design0_n2.pdb	41	3.42	3.14	PROBLEMA	
3. design0_n1.pdb	41	3.76	4.50	PROBLEMA	
4. design0_n29.pdb	41	3.82	2.41	PROBLEMA	
5. design0_n5.pdb	41	4.38	2.58	PROBLEMA	
6. design0_n0.pdb	41	4.65	1.86	PROBLEMA	
7. design0_n15.pdb	41	4.65	1.95	PROBLEMA	
8. design0_n21.pdb	41	4.65	1.96	PROBLEMA	
9. design0_n28.pdb	41	4.65	2.03	PROBLEMA	
10. design0_n3.pdb	41	4.65	1.96	PROBLEMA	
11. design0_n9.pdb	41	4.65	2.03	PROBLEMA	
12. design0_n23.pdb	41	4.69	2.01	PROBLEMA	
13. design0_n18.pdb	41	5.15	2.85	PROBLEMA	
14. design0_n27.pdb	41	5.20	2.04	PROBLEMA	
15. design0_n14.pdb	41	5.38	3.67	PROBLEMA	
16. design0_n22.pdb	41	5.51	3.67	PROBLEMA	
17. design0_n4.pdb	41	5.63	2.58	PROBLEMA	
18. design0_n20.pdb	41	5.81	3.73	PROBLEMA	
19. design0_n30.pdb	41	5.98	6.42	PROBLEMA	
20. design0_n11.pdb	41	6.04	2.56	PROBLEMA	
21. design0_n13.pdb	41	6.17	2.48	PROBLEMA	
22. design0_n26.pdb	41	6.45	3.84	PROBLEMA	
23. design0_n25.pdb	41	6.53	1.85	PROBLEMA	
24. design0_n12.pdb	41	7.48	2.52	PROBLEMA	
25. design0_n31.pdb	41	8.69	4.30	PROBLEMA	
26. design0_n8.pdb	41	11.24	1.41	PROBLEMA	
27. design0_n6.pdb	41	14.78	1.77	PROBLEMA	
28. design0_n17.pdb	41	21.29	2.16	PROBLEMA	
29. design0_n19.pdb	41	23.44	1.43	PROBLEMA	
30. design0_n24.pdb	41	23.44	1.43	PROBLEMA	
31. design0_n7.pdb	41	38.33	1.72	PROBLEMA	
32. design0_n16.pdb	41	48.11	2.24	PROBLEMA	

3. Conclusiones

- Se logró implementar un flujo de trabajo reproducible y aislado mediante Docker.
- RFdiffusion permitió generar geometrías 3D plausibles de proteínas según parámetros definidos (contigs, hotspots).
- ProteinMPNN generó secuencias compatibles con las estructuras diseñadas.
- Rosetta se empleó para evaluar y optimizar la estabilidad estructural.
- PyMOL permitió visualizar y verificar las partes clave del diseño.

Bibliografía

- Jumper, J., Evans, R., Pritzel, A., et al. (2021). *Highly accurate protein structure prediction with AlphaFold*. Nature, 596(7873), 583–589.
- Watson, J. L., Juergens, D., Bennett, N. R., et al. (2023). *De novo design of protein structure and function with RFdiffusion*. bioRxiv. <https://doi.org/10.1101/2023.03.20.533421>
- Anand, N., & Achim, T. (2022). *ProteinMPNN: A deep learning model for protein sequence design*. bioRxiv. <https://doi.org/10.1101/2022.07.20.500902>