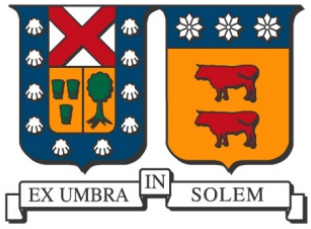
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“ANÁLISIS DE MODELOS DE INTELIGENCIA ARTIFICIAL GENERATIVA EN LA PRODUCCIÓN DE SECUENCIAS DE PROTEÍNAS DE NOVO”

IGNACIO MIGUEL CEA RUIZ

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Dra. Raquel Pezoa Rivera
Profesor Co-Guía: Dr(c). Patricio Flores Herrera
Profesor Correferente: Dr. Roberto León Vásquez

Julio - 2025

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: ANÁLISIS DE MODELOS DE INTELIGENCIA ARTIFICIAL GENERATIVA EN LA PRODUCCIÓN DE SECUENCIAS DE PROTEÍNAS DE NOVO

Nombre del candidato(a): Ignacio Miguel Cea Ruiz

Carrera / Grado: Ingeniería Civil Informática

Campus: Casa Central Valparaiso **;** **Departamento**: Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Raquel Pezoa Rivera, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha:29/07/2025; **Firma:** 

Estudiante o Candidato(a):

Fecha:29/07/2025; **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

A mis padres, Cecilia y Miguel, por haberme formado y cuidado, y por su apoyo incondicional a través de los vaivenes de la vida.

A mi hermana, Geraldine, sin cuyo ejemplo no sería la persona que soy hoy.

A mis sobrinas, Amanda y Dominga, para quienes deseo un futuro más amable y próspero.

Y a la memoria de mis abuelos, porque sin su incansable esfuerzo en vida, nada de esto hubiera sido posible.

AGRADECIMIENTOS

Quiero expresar mis más sinceros agradecimientos a todos quienes hicieron posible, directa o indirectamente, el desarrollo de esta Memoria.

En primer lugar, a mis profesores guía: a la Dra. Raquel Pezoa, por su orientación, apoyo, paciencia y gentileza; y al Dr(c). Patricio Flores, jefe de bioinformática en AQUIT, por confiar en mí para la realización de este proyecto. Agradezco también al equipo de AQUIT por haberme entregado su respaldo en todo lo que necesité para cumplir con la tarea encomendada.

Extiendo mi gratitud a todos los profesores y profesoras que contribuyeron a mi formación profesional y académica, especialmente a los grandes seres humanos que conforman el Departamento de Informática. Mención especial para el profesor Dr. Claudio Torres, quien fue mi profesor guía inicialmente, y para el profesor Dr. Cristián Serpell, para quien fui su ayudante en el curso de Estadística Computacional. También agradezco a la profesora Dra. Pía Amigo, del Departamento de Física y compañera de coro, quien me dio la confianza para exponer sobre este trabajo a sus estudiantes.

A mis compañeros y compañeras de carrera, con quienes compartí en distintos momentos a lo largo de estos años, con quienes salvé tantos ramos solo mediante el poder de la amistad. Su apoyo técnico y moral, y sobre todo moral, fue invaluable.

Más en lo personal, mi más sincero agradecimiento al profesor Felipe Molina, director del Coro de Cámara UTFSM, sin cuya confianza depositada en mí hace ya siete años, cuando me aceptó en dicha agrupación musical, no sé como hubiera sobrevivido a mi tránsito por la carrera. Agradezco también a todos los amigos y amigas en la música que gané en todos estos años, incluyendo a mis amistades del Coro Comunitario de Valparaíso y a su director, el maestro Bernardo Vargas. Sin ustedes, sin la música, sin tantas experiencias compartidas, la vida no sería la misma. Yo no sería el mismo.

Finalmente, gracias a todas mis amistades de la vida, a mis más cercanos, presentes y pasados, a quienes me han acompañado, querido, cuidado y motivado, especialmente durante el proceso de escribir esta Memoria: Charlie, Alvarito, Pato, Carito, Vicente, Pablo y Diego. Gracias por creer en mí cuando yo no podía.

Powered@NLHPC: Esta investigación fue parcialmente apoyada por el National Laboratory for High Performance Computing (NLHPC) de la Universidad de Chile. Se agradece también el apoyo del proyecto ANID PIA/APOYO AFB230003.

RESUMEN

Resumen— Desde hace ya tiempo, la inteligencia artificial se ha consolidado como una herramienta poderosa en bioinformática, posibilitando avances en diversos ámbitos. Más recientemente, se han desarrollado múltiples modelos generativos para abordar el problema del diseño de proteínas *de novo*, diferenciándose claramente entre aquellos que aprenden desde estructuras tridimensionales y aquellos que lo hacen a partir de secuencias de aminoácidos, *i. e.*, los modelos de lenguaje de proteínas. En este trabajo se seleccionaron tres de los mejores modelos de lenguaje de proteínas, se optimizaron para aprender eficazmente desde conjuntos de secuencias pertenecientes a una familia proteica específica y se evaluó su capacidad de generar secuencias de proteínas *de novo* que pertenecieran a dicha familia y fueran estructuralmente viables, destacando ProGen2 como el modelo con mejor desempeño según el algoritmo de evaluación desarrollado. Con ello, se propuso un marco metodológico concreto y replicable para sacar el máximo provecho de estos modelos, priorizando la eficiencia y la interpretabilidad en el diseño de proteínas *de novo*.

Palabras Clave— Inteligencia artificial generativa; Modelos de lenguaje de proteínas; Bioinformática; Diseño de proteínas de novo; Grandes modelos de lenguaje.

ABSTRACT

***Abstract*—** Artificial intelligence has long been established as a powerful tool in bioinformatics, enabling progress in various fields. More recently, numerous generative models have been developed to address the problem of *de novo* protein design, with a clear distinction between those that learn from three-dimensional structures and those that rely on amino acid sequences, *i. e.*, protein language models. In this work, three of the best protein language models were selected, fine-tuned to learn effectively from sets of sequences belonging to a specific protein family, and evaluated in their ability to generate *de novo* protein sequences that belonged to the same family and were structurally viable, with ProGen2 standing out as the best-performing model according to the developed evaluation algorithm. Based on this, a concrete and replicable methodological framework to fully leverage the potential of these models was proposed, with a focus on efficiency and interpretability in *de novo* protein design.

***Keywords*—** Generative artificial intelligence; Protein language models; Bioinformatics; De novo protein design; Large Language Models.

GLOSARIO

ADN: Ácido desoxirribonucleico.
API: *Application Programming Interface* (interfaz de programación de aplicaciones).
ARN: Ácido ribonucleico.
BLAST: *Basic Local Alignment Search Tool* (herramienta básica de búsqueda de alineamiento local).
CDD: *Conserved Domain Database* (base de datos de dominios conservados).
DC: Dominio Conservado.
FASTA: *Fast-All*, estándar para almacenamiento de secuencias de nucleótidos o aminoácidos.
GenAI: *Generative Artificial Intelligence* (inteligencia artificial generativa).
GPU: *Graphics Processing Unit* (unidad de procesamiento gráfico).
HTTP: *HyperText Transfer Protocol* (protocolo de transferencia de hipertexto).
IA: Inteligencia Artificial.
LLM: *Large Language Model* (gran modelo de lenguaje).
LoRA: *Low-Rank Adaptation* (adaptación de bajo rango).
PDB: *Protein Data Bank*, formato de archivo de texto para el almacenamiento de estructuras moleculares tridimensionales.
PEFT: *Parameter-Efficient Fine-Tuning* (ajuste fino eficiente en parámetros).
pLDDT: *predicted Local Distance Difference Test* (prueba de diferencia de distancia local predicha).
PLM: *Protein Language Model* (modelo de lenguaje de proteínas).
RAM: *Random-Access Memory* (memoria de acceso aleatorio).
TPU: *Tensor Processing Unit* (unidad de procesamiento tensorial).
VRAM: *Video* RAM (memoria gráfica de acceso aleatorio).

ÍNDICE DE CONTENIDOS

RESUMEN iv

ABSTRACT iv

GLOSARIO v

ÍNDICE DE FIGURAS viii

ÍNDICE DE TABLAS viii

ÍNDICE DE LISTADOS viii

INTRODUCCIÓN 1

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA 4
1.1 Descripción del problema . . . 5
1.2 Objetivos . . . 7
1.2.1 Objetivo general . . . 7
1.2.2 Objetivos específicos . . . 7

CAPÍTULO 2: MARCO CONCEPTUAL 8
2.1 Inteligencia artificial generativa y aprendizaje profundo . . . 8
2.1.1 Redes neuronales profundas . . . 8
2.2 Grandes modelos de lenguaje . . . 10
2.2.1 Modelos de lenguaje autorregresivos . . . 11
2.2.2 Hiperparámetros en el entrenamiento de un LLM . . . 12
2.2.3 Transformers: *Attention is all you need* . . . 13
2.2.4 Estrategias para muestrear desde un LLM . . . 15
2.2.5 Costos computacionales y optimizaciones . . . 16
2.3 Proteínas . . . 17
2.3.1 Familias de proteínas y dominios conservados . . . 19
2.3.2 Formato FASTA . . . 20
2.3.3 Métricas de similitud de secuencias . . . 21
2.3.4 *Software* para la identificación de dominios conservados . . . 23
2.4 Diseño de proteínas *de novo* . . . 23
2.4.1 Modelos de lenguaje de proteínas para el diseño *de novo* . . . 24

CAPÍTULO 3: PROPUESTA DE SOLUCIÓN 26
3.1 Estudio general del estado del arte . . . 26
3.1.1 Modelos que aprenden de las estructuras de las proteínas: modelos de difusión . . . 26
3.1.2 Modelos que aprenden de las secuencias de las proteínas: PLMs . . . 27
3.2 Selección de PLMs del estado del arte . . . 29

3.2.1 ProtGPT2 . . . 30
3.2.2 ProLLaMA . . . 31
3.2.3 ProGen2 . . . 34
3.2.4 Resumen de los modelos elegidos . . . 36
3.2.5 Pruebas de concepto con los PLMs seleccionados . . . 36
3.3 Diseño experimental . . . 38
3.3.1 Construcción de conjuntos de secuencias para entrenamiento y validación . . . 38
3.3.2 Perfil de dominios de entrenamiento . . . 39
3.3.3 Optimización de hiperparámetros para el ajuste fino . . . 43
3.3.4 Algoritmo de evaluación . . . 45

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN **49**
4.1 Entrenamiento de los modelos . . . 49
4.1.1 Duración del entrenamiento por modelo . . . 49
4.1.2 Comportamiento de las curvas de pérdida . . . 50
4.2 Evaluación de los modelos ajustados . . . 52
4.2.1 Caracterización de las secuencias generadas . . . 52
4.2.2 Resultados de la aplicación del algoritmo de evaluación . . . 54
4.3 Análisis estructural de las secuencias generadas . . . 56
4.3.1 Selección de secuencias para simulación . . . 56
4.3.2 Metodología de simulación estructural y resultados de las simulaciones 58
4.4 Discusión y *benchmarking* . . . 62
4.5 Limitaciones del estudio y propuestas de mejora . . . 63

CAPÍTULO 5: CONCLUSIONES **65**

ANEXOS **67**
5.1 Selección de secuencias para simular estructuras tridimensionales . . . 67

REFERENCIAS BIBLIOGRÁFICAS **70**

ÍNDICE DE FIGURAS

1 Representación de la estructura tridimensional de una proteína, la mioglobina. 2
2 Representación 2D del espacio de secuencias versus una función de aptitud. . 5
3 Ejemplo de red neuronal con dos capas de profundidad. 9
4 Arquitectura *decoder-only* tipo GPT3. 14
5 Estructura general de los aminoácidos. 18
6 Niveles de organización de las proteínas. 19
7 Dominios proteicos y familias de proteínas. 20
8 Alineamiento de secuencias de proteínas histonas de diferentes mamíferos. . 21
9 Tipos de PLMs para la generación de secuencias de proteínas *de novo*. . . . 25
10 Arquitectura general de ProLLaMA. 32
11 Tiempo de entrenamiento por modelo y configuración de hiperparámetros. . 49
12 Curvas de pérdida por configuración de hiperparámetros de ProLLaMA. . . . 50
13 Curvas de pérdida por configuración de hiperparámetros de ProGen2-medium. 51
14 Distribución de largos de las secuencias de entrenamiento. 53
15 Distribución de largos de secuencias generadas por modelo y configuración. . 53
16 Estructuras tridimensionales de transportadores de glucosa naturales. 58
17 Simulaciones estructurales de secuencias generadas por ProtGPT2. 59
18 Simulaciones estructurales de secuencias generadas por ProLLaMA. 60
19 Simulaciones estructurales de secuencias generadas por ProGen2-medium. . 61

ÍNDICE DE TABLAS

1 Cuadro comparativo de las principales características de los PLMs elegidos. . 36
2 Conjuntos de secuencias utilizados para la ejecución de pruebas de concepto. 37
3 Porcentajes de recuperación de dominios conservados en las secuencias generadas. 37
4 Configuraciones de hiperparámetros para el ajuste fino de ProtGPT2. 45
5 Configuraciones de hiperparámetros para el ajuste fino de ProLLaMA. 45
6 Configuraciones de hiperparámetros para el ajuste fino de ProGen2-medium. 45
7 Valores de pérdida finales de los ajustes con cada configuración de ProtGPT2. 52
8 Evaluación de cada configuración de hiperparámetros de ProtGPT2. 54
9 Evaluación de cada configuración de hiperparámetros de ProLLaMA. 55
10 Evaluación de cada configuración de hiperparámetros de ProGen2-medium. . 55
11 Selección de mejores y peores secuencias generadas por cada modelo ajustado. 57

ÍNDICE DE LISTADOS

1 Ejemplo de un registro FASTA de una secuencia de aminoácidos. 21
2 Ejemplo de dos secuencias preprocesadas para el ajuste fino de ProtGPT2. . . 31
3 Ejemplo de conjunto de instrucciones en formato JSON para el ajuste de ProLLaMA. 33

4 Ejemplo de secuencias de proteínas preprocesadas para ProGen2. 35
5 Parámetros de la petición POST a la API de Batch CD-Search y ejemplo de ejecución. 40
6 Ejemplo de contenido de la respuesta de la API de Batch CD-Search al aceptar una petición. 41
7 Ejemplo de extracción de job_id y ejecución de petición GET a la API de Batch CD-Search. 41
8 Ejemplo de respuesta de la API de Batch CD-Search cuando se completa una búsqueda. 42
9 Dos dominios del perfil de dominios de los transportadores de glucosa en el formato JSON correspondiente. 43
10 Mejores y peores secuencias generadas por ProtGPT2 - Configuración B. . . . 67
11 Mejores y peores secuencias generadas por ProLLaMA - Configuración C. . . 68
12 Mejores y peores secuencias generadas por ProGen2-medium - Config. D. . . 69

INTRODUCCIÓN

En los últimos años, las aplicaciones de la inteligencia artificial generativa (GenAI, por sus siglas en inglés) se han tomado las conversaciones en todos los campos del conocimiento y en industrias de todas las áreas. La popularización de modelos de inteligencia artificial (IA) basados en aprendizaje profundo que son capaces de generar todo tipo de contenido según las instrucciones del usuario ha iniciado una nueva era en la forma en que las personas trabajan, estudian y viven su vida en general. Algunos autores comparan este hito con la masificación del Internet y de los teléfonos inteligentes [Ooi *et al.*, 2025]. Uno de los campos del conocimiento que se ha visto revolucionado con esta tecnología es la bioinformática, y es esta el área dentro de la cual se enmarca este trabajo, el cual surge a partir de una propuesta realizada por **AQUIT, una empresa emergente dedicada a la investigación biotecnológica para una acuicultura sustentable**.

Es fácil creer que el estudio de la bioinformática, también llamada «biología computacional», es relativamente nuevo. Sin embargo, las primeras aplicaciones de métodos computacionales a la biología ocurrieron en la década de 1960 y, de hecho, involucraron el ensamblaje de secuencias de proteínas *de novo* [Gauthier *et al.*, 2019]. Con el paso de las décadas, muchos avances en la biología se lograron gracias a la bioinformática, incluyendo hitos que «lo cambiaron todo» como la secuenciación del genoma humano [Gibbs, 2020].

El **foco de esta memoria**, como lo indica su título, está sobre modelos de GenAI que producen secuencias de proteínas *de novo*. Las proteínas son parte fundamental de la mecánica biológica subyacente a todos los seres vivos conocidos, estando codificadas en su propio material genético y llevando a cabo la mayoría de las funciones críticas de las células: formación de tejidos, procesos de crecimiento y desarrollo, metabolismo, transporte de oxígeno y nutrientes, entre muchas otras. Una proteína es una macromolécula compuesta por una o más secuencias lineales de aminoácidos (de ahí que se hable de «secuencias» de proteínas) que se pliegan sobre sí mismas debido a sus propiedades fisicoquímicas, adquiriendo cualidades y funciones específicas. La mayoría de los seres vivos cuentan con veinte tipos de aminoácidos, de los cuales una proteína puede llegar a estar conformada comúnmente por decenas a decenas de miles, pudiendo aparecer en cualquier orden y repetición.

El **diseño de proteínas *de novo*** es el diseño desde cero de secuencias de proteínas nuevas, nunca antes vistas en la naturaleza, que tengan estructuras y funcionalidades específicas [Richardson y Richardson, 1989]. Lamentablemente, el espacio de búsqueda para diseñar proteínas *de novo* es astronómico [González-Torres *et al.*, 2007], *e.g.*, para una secuencia de 100 aminoácidos, existirían 20^{100} posibles combinaciones. Además, cuando una proteína termina de ensamblarse, se pliega sobre sí misma y encuentra naturalmente, a veces en cuestión de segundos, su configuración tridimensional de mínima energía (ver ejemplo en la Figura 1), fenómeno conocido como la paradoja de Levinthal [Nelson *et al.*, 2019]. El problema de predecir con certeza esa configuración final se denomina «problema de plegamiento de proteínas» y por muchos años había sido considerado intratable en la práctica.

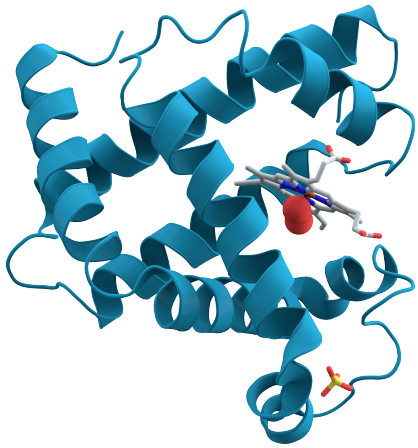


Figura 1: Representación de la estructura tridimensional de una proteína, la mioglobina. Fuente: Simulación a partir de entrada PDB vía Wikimedia Commons.

Sin embargo, hace cinco años, se consideró resuelto este problema con el lanzamiento de AlphaFold2 [Heaven, 2020], un modelo computacional de aprendizaje profundo capaz de resolver con alto desempeño el problema de plegamiento de proteínas, basándose solamente en su secuencia de aminoácidos. Este hito fue considerado una revolución, e incluso «el logro más importante en IA hasta ahora» [Toews, 2021]. Desde entonces, el entusiasmo por lo que la IA puede ser capaz de lograr en biología molecular no ha dejado de crecer, dando lugar a importantes desarrollos desde, en particular, los modelos de GenAI. Algunos de ellos procesan información de las estructuras tridimensionales de las proteínas, mientras que otros procesan las secuencias de aminoácidos, las cuales pueden ser abordadas por los grandes modelos de lenguaje (LLMs, por sus siglas en inglés).

El potencial de los LLMs radica en su capacidad de aprender el «lenguaje» de las proteínas, infiriendo patrones y estructuras para generar salidas completamente novedosas, pudiendo diseñar proteínas *de novo* eficazmente. La naturaleza combinatoria de las proteínas, sumado a la idea de que la parte del espacio de proteínas explorado por las fuerzas de la evolución durante millones de años es solo una pequeña fracción del total [Toews, 2023], es un terreno sumamente fértil para el desarrollo de LLMs especializados, los cuales pueden devenir en todo tipo de aplicaciones en medicina, biotecnología, acuicultura, agricultura, etc. Además, existe la posibilidad de darle bases a una eventual «ciencia teórica de las proteínas», tal como explica el profesor de biología molecular de la Universidad de Columbia, Dr. Mohammed AlQuraishi:

> *Lo que para mí es más apasionante de esta idea es el surgimiento de una capa de fenómeno biológico que tiene sus raíces en la física* [...], *pero que puede describirse independientemente de ella. Es decir, las primitivas de este fenómeno estarían lo suficientemente abstraídas de la física subyacente como para que no necesitemos de la física para modelarlo. Podemos simplemente operar en el*

> *nivel de fragmentos y motivos de proteínas, construyendo una gramática probabilística que describa cómo estas partes se combinan e interactúan, sin recurrir nunca a una simulación de fuerza bruta* [...]. *En última instancia, tales simulaciones solo serían ejercicios para hacer física realmente bien y a escala. Pero la posibilidad de que exista una descripción del espacio de la estructura de las proteínas que pueda formularse sin recurrir a la física subyacente* ***es la posibilidad de una ciencia teórica de las proteínas****, independiente de* [...] *la teoría física.*[1]
>
> [AlQuraishi, 2018]

En el Capítulo 1 se define el problema central que llevó a la necesidad de analizar en profundidad la eficacia de los modelos de GenAI en la producción de secuencias de proteínas *de novo*, con foco particular sobre LLMs, junto a la delimitación del alcance de este trabajo y la presentación de sus objetivos. En el Capítulo 2 se expone el marco conceptual, profundizando en los conceptos biológicos y bioinformáticos relevantes, incluyendo el diseño de proteínas *de novo*, así como describiendo en detalle los fundamentos teóricos de la IA generativa y de los LLMs. En el Capítulo 3 se propone la solución al problema definido, describiendo el enfoque metodológico elegido para la implementación y evaluación de los modelos, detallando las configuraciones de *hardware*, conjuntos de datos y algoritmos utilizados. En el Capítulo 4 se expone la validación de la solución propuesta, presentando los resultados obtenidos y una discusión respectiva. Finalmente, en el Capítulo 5 se presentan las conclusiones del trabajo realizado.

[1] Traducción propia.

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

El **diseño de secuencias de proteínas *de novo***, como se mencionó en la Introducción, es una subárea de la bioinformática de ya larga data. Como tal, es uno de dos acercamientos al problema del **diseño racional de proteínas**, siendo el otro el **rediseño de proteínas** (*i.e.*, desarrollar variantes calculadas a partir de una proteína ya conocida), el cual ha mostrado numerosos casos de éxito por sobre el diseño *de novo*, debido a las limitaciones computacionales históricas de este último [Kortemme, 2024]. En particular, «diseño racional» se define como la estrategia de crear nuevas proteínas basándose en la habilidad de predecir funcionalidades y estructuras deseadas; su contraparte es la estrategia de la «evolución dirigida», la cual consiste en la construcción de nuevas secuencias a partir de ciclos iterativos de mutaciones aleatorias, siguiendo diferentes métodos [Song *et al.*, 2023].

Según otros autores, se pueden definir tres enfoques históricos distintos en el diseño de proteínas, en orden cronológico: el minimalista, el racional y el **computacional** [Woolfson, 2021]. Este último, tradicionalmente, se ha caracterizado por iniciar definiendo una estructura tridimensional que se busca lograr, recurriendo a simulaciones físicas a nivel atómico para, luego, encontrar una secuencia de aminoácidos que se pliegue a esa estructura. En otras palabras, el diseño de proteínas computacional ha sido definido típicamente como un problema de optimización matemática: «Dada una estructura y función definidas por el usuario, encuentre una o más secuencias de aminoácidos de baja energía que adopten de forma estable la estructura deseada y realicen la función definida» [Pan y Kortemme, 2021], donde un algoritmo (*e.g.*, Monte Carlo) busca el mínimo global de una función de energía multivariada [Ferruz *et al.*, 2023], lo cual tiene sus limitaciones, tales como el hecho de que una desviación de tan solo un ángstrom basta para que un diseño falle (ver Figura 2). Sin embargo, los desarrollos en IA y modelos de aprendizaje profundo de los últimos años han dado lugar al nacimiento de **un enfoque generativo para el diseño *de novo***, el cual tiene varias ventajas, tales como el descubrimiento de funciones que posiblemente aún no se han visto en la naturaleza, así como la aplicación de principios de ingeniería en la generación de secuencias de proteínas [Kortemme, 2024].

Así, aprovechando el desarrollo de la GenAI, algunos trabajos de diseño de proteínas han adoptado un enfoque basado en la estructura como punto de partida para diseñar nuevos plegamientos y proteínas [Anand *et al.*, 2022, Dauparas *et al.*, 2022], mientras que otros se han centrado en la **secuencia de aminoácidos**, entrenando grandes modelos generativos de lenguaje a partir de extensas bases de datos de secuencias [Ferruz *et al.*, 2022, Nijkamp *et al.*, 2023]. Si bien el progreso en este campo depende de la integración de diferentes enfoques, como los mencionados, o incluso sobre la función final, ya que ninguno por sí solo puede explicar completamente los fenómenos celulares necesarios para el diseño de proteínas, en el presente trabajo el foco estará en el análisis de **grandes modelos de lenguaje que trabajan con secuencias de aminoácidos**.

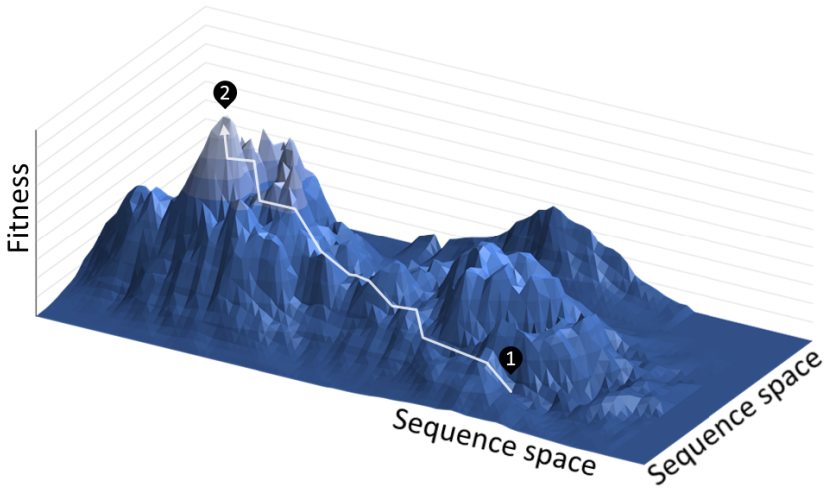


Figura 2: Representación 2D del espacio de secuencias versus una función de aptitud. Los enfoques tradicionales de diseño computacional de proteínas aplican algoritmos que exploran el espacio de secuencias de aminoácidos (*sequence space)* en busca de aquellas que optimicen una función de aptitud (*fitness function*). La selección natural ha hecho lo mismo a lo largo de millones de años, optimizando proteínas desde una aptitud basal (1) a una aptitud óptima (2) generación tras generación. Fuente: Extraído de [Shafee, 2014].

Entre los ya numerosos trabajos que han buscado aplicar técnicas de GenAI en la producción de secuencias de proteínas *de novo*, así como en otras tareas afines, existen autores que han definido lo que llaman **modelos de lenguaje de proteínas** (PLMs, por sus siglas en inglés) [Bepler y Berger, 2021, Ofer *et al.*, 2021]: LLMs especializados en abordar el problema del «procesamiento del lenguaje de las proteínas» [Lv *et al.*, 2024]. La idea general es que, dado que las estructuras primarias de las proteínas son sus secuencias de aminoácidos y que estas se pueden codificar como cadenas de caracteres, el lenguaje de proteínas subyacente, con todos sus patrones y motivos, es altamente aprendible por los LLMs del estado del arte. Luego, un LLM convertido en un PLM podría, en teoría, ser capaz de generar proteínas *de novo* funcionales, explorando rincones del espacio de secuencias de proteínas (y, posiblemente, de estructuras y funciones) no explorados por la naturaleza.

1.1. Descripción del problema

Como se mencionó anteriormente, se han desarrollado diversos LLMs para el procesamiento del lenguaje de las proteínas. Sin embargo, los trabajos que han dado origen a dichos modelos son, en general, muy nuevos, por lo que la existencia de modelos de *benchmarking* que permitan decir con cierto nivel de certeza qué modelos son más óptimos que otros y bajo qué escenarios es escasa. La mayoría del tiempo la implementación de estos modelos para la tarea de diseñar proteínas *de novo* no es directa, ya sea por problemas en la documentación o en el acceso, lo cual genera una barrera para científicos y profesionales del área que no cuenten con las herramientas necesarias para lograr sus objetivos.

Otro problema observado tiene que ver con el tono utilizado en los artículos que presentan nuevos modelos que generan proteínas con IA, los cuales suelen ser descritos con grandilocuencia por parte de sus autores. Si bien la mayoría de estas publicaciones ya cuentan con una validación académica rigurosa, su adopción práctica por parte de usuarios no especializados en IA suele presentar dificultades significativas. Por ejemplo, los investigadores que desarrollaron el modelo ProGen afirmaron que este era capaz de generar proteínas altamente optimizadas en cuestión de segundos, las que podrían incluso llegar a rivalizar con proteínas naturales, las cuales han requerido millones de años de evolución para obtener sus formas actuales [Madani *et al.*, 2023]. Sin embargo, al intentar implementar estos modelos en escenarios reales, aparecen obstáculos como barreras de licenciamiento, la generación de secuencias de proteínas incoherentes al variar algunos parámetros o la falta de documentación. Estas dificultades no solo obstaculizan la aplicación de estos modelos, sino que también siembran dudas respecto a su efectividad y a las afirmaciones de sus desarrolladores.

Además de lo anterior, existen desafíos asociados a la validación experimental de las secuencias generadas, los cuales añaden una capa de complejidad más al proceso de diseño de proteínas *de novo*. El mayor desafío consiste en predecir computacionalmente la funcionalidad de las secuencias, ya que la síntesis en laboratorio es costosa: por ejemplo, en este trabajo se generaron secuencias de 526 aminoácidos de largo en promedio; luego, producir una secuencia *de novo* de ese largo costaría alrededor de USD$2.000[2]. Entonces, si no existe certeza respecto a la efectividad de algún PLM dado en su capacidad de generar proteínas funcionales, producirlas en laboratorio se vuelve sumamente riesgoso desde un punto de vista financiero, lo cual merma la confianza de los biólogos y especialistas en la IA para esta tarea.

Por ello, desde empresas emergentes como **AQUIT**, quienes propusieron este trabajo, se plantea como problemática principal la **falta de herramientas o trabajos que permitan analizar y comparar exhaustivamente modelos de IA que producen secuencias de proteínas *de novo***, de forma que se puedan utilizar efectiva y eficientemente para promover la innovación biotecnológica, tomando en consideración alcances de tiempo y costo computacional, así como condiciones óptimas de ejecución (cantidad de secuencias de entrenamiento, hiperparámetros de los modelos, requerimientos de *hardware*, entre otros). Esta memoria busca ser un aporte en esta problemática mediante el análisis exhaustivo de modelos de IA de interés presentes en el estado del arte, evaluando su efectividad y determinando el alcance de cada uno de ellos. Los resultados de este trabajo permitirán mitigar riesgos tales como la adopción de modelos de IA inadecuados, el desaprovechamiento de su potencial innovador y la pérdida de confianza en la IA.

Adicionalmente, cabe destacar algunos de los alcances que podrían llegar a tener estos modelos, si es que realmente son capaces de explorar eficientemente zonas desconocidas del espacio latente de proteínas: nuevos y mejores antibióticos, tratamientos para enfermeda-

[2]Se considera el costo de sintetizar el gen artificial para la producción de la secuencia deseada, procesos de purificación y costos de servicio asociados, según datos de `https://www.abiscientific.com/peptide_synthesis_price.html` y `https://www.genscript.com/gene_synthesis.html`.

des raras, una producción de alimentos más sostenible y, como se mencionó en la Introducción, un mejor entendimiento de la mecánica subyacente a la formación de las estructuras tridimensionales y funcionalidades que caracterizan a las proteínas.

1.2. Objetivos

1.2.1. Objetivo general

Analizar modelos de inteligencia artificial generativa enfocados en la producción de secuencias de proteínas *de novo*, pertenecientes al estado del arte, y evaluar su efectividad en base a métricas que permitan cuantificar computacionalmente la calidad de las secuencias generadas.

1.2.2. Objetivos específicos

O1. Precisar el estado del arte respectivo a modelos de IA generadores de secuencias de proteínas *de novo*.

O2. Definir computacionalmente las métricas bioquímicas necesarias y, en base a ellas, diseñar un algoritmo que permita evaluar la calidad de los modelos de IA y de las secuencias de proteínas que generan.

O3. Implementar correctamente cada uno de los modelos de IA del estado del arte elegidos y evaluarlos según corresponda.

O4. Determinar, mediante un *benchmarking*, la plataforma más efectiva para la generación de secuencias de proteínas *de novo*, evaluando también el alcance de cada una.

CAPÍTULO 2
MARCO CONCEPTUAL

El presente capítulo tiene por objetivo la introducción de los principales conceptos teóricos y tecnológicos que sustentan el posterior análisis de modelos de IA generativa, en particular LLMs, en la producción de secuencias de proteínas *de novo*. Estos conceptos proveen el contexto necesario para comprender la motivación, el enfoque de la solución propuesta y la interpretación de los resultados obtenidos.

2.1. Inteligencia artificial generativa y aprendizaje profundo

La **inteligencia artificial generativa** (GenAI, por sus siglas en inglés) es un concepto amplio que agrupa a todas aquellas técnicas basadas en IA que se enfocan en la **generación de contenido**. Actualmente, este contenido puede variar desde texto escrito hasta salidas más complejas como imágenes, audio y vídeos.

Históricamente, ha existido una clara distinción entre modelos de IA **discriminativos** y **generativos**. Una analogía común es que entrenar un modelo discriminativo es como enseñarle a un niño muchas imágenes de perros y gatos para que, luego, al ver una nueva imagen, pueda decidir a qué categoría corresponde; mientras que, por otro lado, entrenar un modelo generativo es similar, pero, en vez de lo anterior, se le pide al niño que dibuje un perro o un gato basándose en lo que ha visto. Los modelos generativos, en principio, se clasifican como **aprendizaje no supervisado**, ya que aprenden la distribución de probabilidad subyacente a los datos, sin que estos vengan acompañados de una etiqueta específica.

2.1.1. Redes neuronales profundas

Actualmente, la tecnología detrás del gran éxito que han tenido los modelos de GenAI son las **redes neuronales profundas**. Una red neuronal profunda es una serie de múltiples capas de «neuronas»: unidades básicas de computación que reciben una entrada **x**, calculan una combinación lineal de cada dimensión de la entrada y generan una salida **y**, luego de aplicar una **función de activación** que determina la intensidad de la salida (ver ejemplo en la Figura 3). Las redes neuronales profundas modernas pueden llegar a contener más de cien capas ocultas (*i. e.*, la «profundidad» de la red) con miles de neuronas cada una (*i. e.*, el «ancho» de la red), donde los coeficientes o «pesos» utilizados en cada combinación lineal a calcular son los **parámetros** del modelo.

Un modelo de **aprendizaje profundo** es aquel que utiliza una arquitectura de red neuronal profunda para aprender de un conjunto de datos, ya sea de forma supervisada o no super-

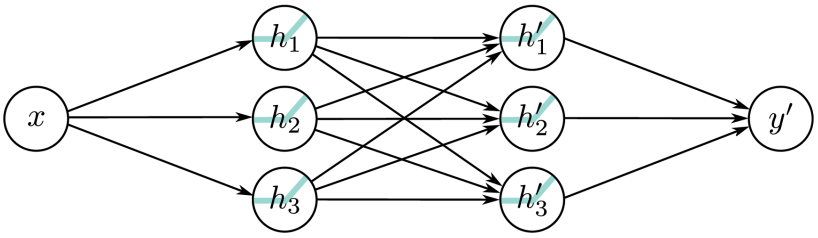


Figura 3: Ejemplo de red neuronal con dos capas de profundidad.
Cada capa contiene tres unidades ocultas o «neuronas». Fuente: Extraída de [Prince, 2023].

visada. Existen distintas arquitecturas en la literatura actual para el aprendizaje de variados tipos de tareas, pero todas tienen en común la utilización de una **función de pérdida**, la cual cuantifica la diferencia entre la salida actual producida por la red y la salida esperada, y algún **algoritmo de optimización** para encontrar el conjunto de valores de los parámetros que **minimiza** el valor de la función.

Para minimizar el valor de la función de pérdida, *i. e.*, que la red «aprenda», se utiliza un **conjunto de datos de entrenamiento**. Cada ejemplo de entrenamiento atraviesa la red, obteniendo un valor conocido como *train loss* utilizando la función de pérdida. Luego, el algoritmo de optimización intenta modificar los pesos de la red en la dirección que disminuye la *train loss*. Cuando la red ha visto todos los ejemplos de entrenamiento, se utiliza un **conjunto de datos de validación** independiente para evaluar el desempeño actual de la red, utilizando la misma función de pérdida para obtener un valor conocido como *validation/test loss*, pero sin propagar cambios a los pesos. Este ciclo de entrenamiento y validación se repite un determinado número de veces y, en algunos casos, se realiza una evaluación del modelo final con otro conjunto independiente llamado **conjunto de datos de prueba**. En la literatura no existe un acuerdo sobre la diferencia entre los conceptos de validación y prueba, pero en este trabajo se denominará «validación» al conjunto apartado para evaluar el desempeño de la red entre cada ciclo, mientras que la evaluación final se realizará siguiendo un procedimiento a explicar en el Capítulo 3.

Los algoritmos que optimizan la función de pérdida se basan en el algoritmo de ***backpropagation***[3] para calcular las derivadas de la función de pérdida al final de cada ciclo de entrenamiento. Una de los primeros algoritmos de optimización fue el **descenso de gradiente estocástico**. Su implementación más utilizada hoy en día es el algoritmo **Adam** (del inglés *Adaptive moment estimation*, *i. e.*, «estimación de momento adaptativo»), el cual utiliza dos hiperparámetros adicionales para desarrollar un método que ajusta los parámetros del modelo eficientemente, tomando en cuenta el historial del gradiente de cada uno, permitiendo una convergencia más suave hacia los parámetros óptimos [Prince, 2023]. Una variante ampliamente utilizada en contextos donde la regularización es clave es **AdamW**, que modifica el término de decaimiento de pesos para mejorar la generalización del modelo [Loshchilov y Hutter, 2017].

[3] «Retropropagación» en español, aunque poco usado.

Todos los elementos anteriores dotan a los modelos generativos de la capacidad de aprender la distribución de probabilidad de los datos a tal punto que, luego, al recibir una entrada específica, pueden generar salidas novedosas y coherentes con base en las distribuciones aprendidas. Por ejemplo, un modelo entrenado sobre textos en español aprenderá que ciertas palabras tienen mayor probabilidad de seguir a otras en un texto, que ciertas estructuras o estilos son más preferibles a otros dependiendo del contexto, entre otros elementos que, luego, podrá utilizar para producir contenido nuevo y coherente. Este mismo acercamiento es particularmente aplicable al dominio de las proteínas, como se ha mencionado anteriormente. Por ello, los modelos de inteligencia artificial generativa analizados en este trabajo corresponden a grandes modelos de lenguaje.

2.2. Grandes modelos de lenguaje

Un **gran modelo de lenguaje** (LLM, por sus siglas en inglés), también llamado **modelo extenso de lenguaje**, es una arquitectura de aprendizaje profundo usualmente clasificada como GenAI que es entrenada para aprender los patrones y características del «lenguaje natural» (*i. e.*, el lenguaje hablado por los humanos) a partir de un gran conjunto de datos. Estos modelos pueden realizar multitud de tareas al recibir *prompts* o entradas en lenguaje natural y ya son ampliamente utilizados como *chatbots*, agentes, asistentes para programar, estudiar y crear contenido personalizado en general [Sandhu, 2024]. Por otro lado, también han permitido estudiar por analogía fenómenos emergentes como el lenguaje mismo y el razonamiento. Actualmente, su gran desempeño se debe casi en su totalidad al **mecanismo de atención** y a la arquitectura de redes neuronales profundas conocida como ***transformers*** [Bahdanau, 2014, Vaswani *et al.*, 2017]. El sistema más exitoso de este tipo hasta la fecha ha sido la serie GPT de OpenAI; su plataforma en línea, ChatGPT, ganó más de cien millones de usuarios en solo dos meses desde su lanzamiento en noviembre de 2022, el crecimiento más rápido de la cantidad de usuarios de alguna aplicación en la historia [Milmo, 2023].

Lo que hace «extensos» a estos modelos de lenguaje es la cantidad de parámetros que contienen sus redes neuronales profundas subyacentes, alcanzando números del orden de las decenas a centenas de miles de millones. No existe una regla o acuerdo universal al respecto, pero algunos expertos han propuesto que, para que un modelo de lenguaje sea extenso, este debe contener al menos 10 mil millones de parámetros [Zhao *et al.*, 2025]. De todas formas, se requieren recursos computacionales significativos para ejecutar modelos de ese tamaño, por lo que en este trabajo se limitó el análisis a modelos de menor tamaño, pero que igualmente pueden denominarse LLMs debido a su arquitectura.

2.2.1. Modelos de lenguaje autorregresivos

La mayoría de los LLMs, incluyendo aquellos analizados en este trabajo, son modelos de lenguaje **autorregresivos**[4]. Su principal característica es la predicción de la siguiente palabra o *token* en una secuencia en base a las palabras previas dadas como *prompt*. Formalmente, dado un conjunto de *tokens* en una secuencia $t_1, t_2, \ldots, t_N$ (*i. e.*, un trozo de texto), un modelo autorregresivo aprende la distribución de probabilidad conjunta de la secuencia según:

$$\mathrm{P}(t_1, t_2, \ldots, t_N) = \prod_{n=1}^{N} \mathrm{P}(t_n | t_1, \ldots, t_{n-1}) \tag{1}$$

Este enfoque permite que los modelos generen secuencias coherentes y contextualmente relevantes, ya que cada *token* generado depende de todos los *tokens* anteriores [Prince, 2023].

La definición anterior se utiliza para encontrar una función de pérdida adecuada. En particular, al entrenar un modelo autorregresivo se sigue el principio de máxima verosimilitud, *i. e.*, se busca encontrar **un conjunto de parámetros óptimos $\hat{\theta}$ tal que la probabilidad conjunta de cada secuencia real se maximice**, es decir:

$$\hat{\theta} = \arg\max_{\theta} \left[\mathrm{P}(t_1, \ldots, t_N | \theta)\right] = \arg\max_{\theta} \left[\prod_{n=1}^{N} \mathrm{P}(t_n | t_1, \ldots, t_{n-1}; \theta)\right] \tag{2}$$

Pero la productoria de probabilidades es numéricamente inestable, por lo que se trabaja con la suma de sus logaritmos, que es equivalente:

$$\hat{\theta} = \arg\max_{\theta} \left[\sum_{n=1}^{N} \log \mathrm{P}(t_n | t_1, \ldots, t_{n-1}; \theta)\right] \tag{3}$$

La expresión entre corchetes en la ecuación anterior se puede transformar en una función de pérdida al multiplicar por (-1), obteniendo así una función tal conocida como **entropía cruzada** (o *cross-entropy* en inglés):

$$\hat{\theta} = \arg\min_{\theta} \underbrace{\left[-\sum_{n=1}^{N} \log \mathrm{P}(t_n | t_1, \ldots, t_{n-1}; \theta)\right]}_{\text{Entropía cruzada}} \tag{4}$$

La entropía cruzada es una medida de la incertidumbre en la diferencia entre una distribución de probabilidad real de cada *token* y una distribución modelada [Prince, 2023]. Al ser utilizada para el entrenamiento de un LLM, se dice que su objetivo es el **modelamiento causal del lenguaje** [Lv *et al.*, 2024].

[4] El término «autorregresivo» proviene originalmente del análisis de series temporales en estadística y econometría [Kendall, 1952], siendo bastante posterior su adaptación al ámbito del procesamiento de lenguaje natural.

Una vez entrenado un LLM autorregresivo, se puede tomar una secuencia de *tokens* generada por él y utilizar el modelo para calcular su **perplejidad**, *i. e.*, una medida de la incertidumbre presente en las predicciones de un modelo. Se calcula como la exponencial de la entropía cruzada de la secuencia (ver ecuación 5) y se interpreta como la cantidad de *tokens* de entre los cuales está eligiendo el modelo para predecir cada *token* siguiente. Por ello, una baja perplejidad indica que el modelo está más seguro respecto a sus decisiones al generar la secuencia en cuestión, es decir, está menos *perplejo* [Das Gupta, 2025].

$$\text{Perplejidad}(t_1, \ldots, t_N) = \exp\left[-\sum_{n=1}^{N} \log \mathrm{P}(t_n | t_1, \ldots, t_{n-1}; \theta)\right] \tag{5}$$

2.2.2. Hiperparámetros en el entrenamiento de un LLM

En la subsección 2.1.1 se mencionaron los conceptos de profundidad y ancho de una red neuronal profunda. Ellos corresponden a ejemplos de **hiperparámetros** de un modelo, los cuales son cantidades que se definen antes de aprender los parámetros del modelo y afectan el alcance y el desempeño de dicho aprendizaje. No existen reglas claras para la elección de hiperparámetros óptimos, por lo que es una práctica común entrenar el mismo modelo varias veces con distintas combinaciones de hiperparámetros hasta encontrar la que dé un mejor desempeño en el modelo final [Prince, 2023].

Un hiperparámetro relevante es el ***batch size*** o tamaño de lote. Al entrenar un LLM, se subdivide el conjunto de secuencias de entrenamiento en un número fijo de *batches* (palabra inglesa para «lotes») aleatorios de secuencias: todas las secuencias de un *batch* son procesadas de forma simultánea por el modelo. A mayor tamaño de cada *batch* el modelo tendrá mayor contexto a su disposición, por lo que podrá converger más rápido. Sin embargo, existe un compromiso entre el tamaño de *batch* y los recursos computacionales disponibles, por lo que este hiperparámetro debe ser optimizado.

Por otro lado, la **tasa de aprendizaje**, o *learning rate*, también es de gran importancia, ya que define la «velocidad» con la que el modelo convergerá o no a un conjunto de parámetros óptimos. En particular, el gradiente de la función de pérdida es el operador matemático que le indica al modelo cuánto debe cambiar sus parámetros en cada iteración para acercarse al óptimo; la tasa de aprendizaje controla la magnitud de esos cambios. Una tasa de aprendizaje pequeña puede estancar al modelo en óptimos locales, mientras que una grande puede saltarse los óptimos. Por ello, se debe elegir con mucho cuidado.

Muchos algoritmos implementan una tasa de aprendizaje planificada, de tal forma que parten con un valor alto para ayudar al modelo a explorar el espacio de parámetros, para luego disminuirlo a un valor más fijo con el objetivo de ir afinando los pesos. Una alternativa es el uso de ***warmup steps***, *i. e.*, se define un número de pasos iniciales en los que la tasa crece gradualmente desde cero hasta un valor deseado, a partir del cual se sigue una planifica-

ción definida por el resto del entrenamiento; esto evita inestabilidades al inicio del entrenamiento [Kalra y Barkeshli, 2024]. Por otro lado, algoritmos de optimización como Adam introducen hiperparámetros adicionales que deben ser optimizados también [Prince, 2023].

Un ciclo en el cual el modelo procesa todos los *batches* en que se subdividió el total de las secuencias de entrenamiento y modifica sus pesos acorde al gradiente de la función de pérdida y su tasa de aprendizaje correspondiente se denomina ***epoch***[5]. Utilizar demasiados *epochs* puede producir sobreajuste en el modelo (*i. e.*, memorización de las secuencias de entrenamiento y poca capacidad de generalizar), mientras que muy pocos, dependiendo del tamaño del conjunto de entrenamiento, pueden provocar subajuste (*i. e.*, incapacidad de obtener un aprendizaje relevante de las secuencias de entrenamiento).

Existen, naturalmente, otros hiperparámetros que afectan el desempeño final de un modelo. El tamaño de los conjuntos de entrenamiento y validación es uno de los más importantes, existiendo relaciones de dependencia con hiperparámetros como los mencionados anteriormente, pero a veces no se tiene total control sobre ellos. Por otro lado, la arquitectura de un LLM puede traer consigo otros hiperparámetros particulares que también deben ser optimizados para mejorar el desempeño del modelo, complejizando el espacio de búsqueda.

2.2.3. Transformers: *Attention is all you need*

Al inicio de esta sección se mencionó que la arquitectura de LLMs conocida como ***transformers*** fue crucial para el desarrollo de modelos de lenguaje exitosos. En su origen, su principal aplicación era la traducción entre diferentes lenguajes humanos, pero hoy en día encuentra varias otras aplicaciones. Ejemplos relevantes son las conversiones entre audio y texto, texto e imágenes, así como el análisis de lenguajes con patrones desconocidos, como el de las proteínas. Sus autores demostraron que un modelo de lenguaje solo necesitaba el **mecanismo de atención** para obtener buenos resultados, dejando obsoletos modelos anteriores basados en redes neuronales recurrentes o convolucionales [Vaswani *et al.*, 2017].

La motivación detrás del concepto de «atención» se puede entender con el siguiente ejemplo:

> «El restaurante se negó a servirme un sándwich de jamón porque solo cocina comida vegetariana. Al final, solo me dieron dos rebanadas de pan. Su ambiente era tan bueno como su comida y su servicio.»[6]

Un buen modelo de lenguaje debe ser capaz de determinar si dicha frase es positiva o negativa o responder a preguntas como «¿El restaurante sirve bistec?». Para ello, debe poder

[5] Palabra prestada del griego *epochē*, literalmente «pausa» o «detención», con el significado filosófico de un momento de reflexión en torno a algo que se considera o examina. Ver `https://www.britannica.com/topic/epoche`.

[6] Ejemplo adaptado de [Prince, 2023].

codificar la información de una forma que no sea computacionalmente intensiva, adaptarse a entradas de diferente largo y, especialmente, entender sutilezas semánticas como el hecho de que «solo cocina» y «me dieron» hacen referencia (*i. e.*, ponen *atención*) al sujeto «el restaurante».

La arquitectura original de un *transfomer* incluye una capa de codificación (*encoder*) para obtener una representación interna del *prompt*, tal que capture su significado y contexto, y otra de decodificación (*decoder*) que se encarga de aprender a generar nuevos *tokens* de forma autorregresiva. Los LLMs que generan proteínas analizados utilizan **arquitecturas *decoder-only***, ya que su interés está solo en el modelamiento causal del lenguaje (ver ejemplo en Figura 4) [Hrbáň y Hoksza, 2024].

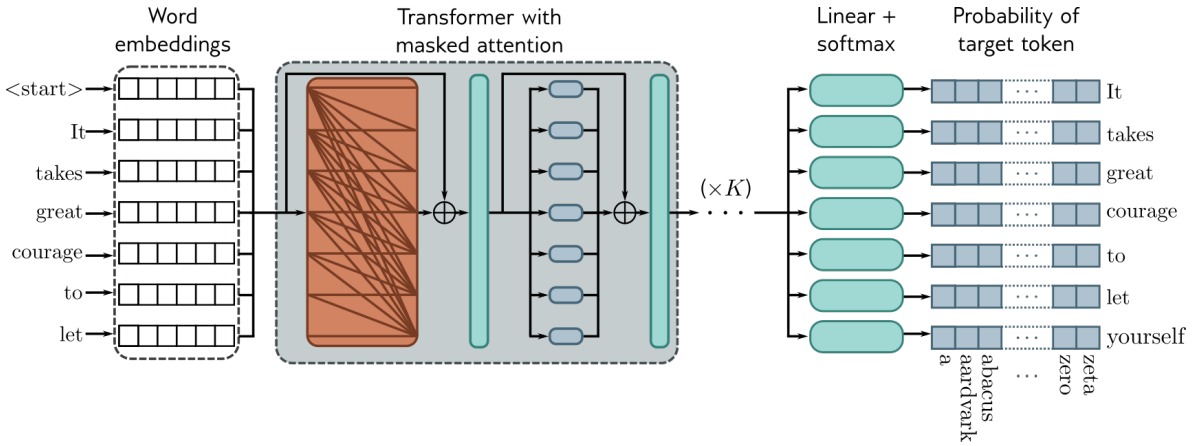


Figura 4: Arquitectura *decoder-only* tipo GPT3.
Los *tokens* son transformados a *word embeddings*, los cuales atraviesan una serie de capas *transformer* que usan atención enmascarada: cada posición en la secuencia solo puede *atender* a su propio *embedding* o el de los *tokens* precedentes, con el objetivo de maximizar la probabilidad modelada del siguiente *token* en la secuencia, lo cual corresponde a una tarea autorregresiva. Fuente: Extraída de [Prince, 2023].

El primer paso en un modelo que usa *transformers* es la construcción de ***word embeddings***[7] a partir de los *tokens* de entrada. Cada *transformer* cuenta con un *tokenizador*, el cual se encarga de dividir la entrada según un «vocabulario» $\mathcal{V}$ de posibles *tokens*, y una matriz de *embeddings* $\mathbf{\Omega}_e \in \mathbb{R}^{D\times|\mathcal{V}|}$, siendo D el *embedding size* o número de dimensiones en las cuales se proyectan los *tokens*. De esta forma, los *word embeddings* se calculan como $\mathbf{X} = \mathbf{\Omega}_e\mathbf{T}$, siendo $\mathbf{T} \in \mathbb{R}^{|\mathcal{V}|\times N}$ una matriz donde la n-ésima columna corresponde al n-ésimo token y contiene un uno en la casilla del *token* correspondiente en el vocabulario y ceros en todas las otras casilas. Los parámetros de $\mathbf{\Omega}_e$ se aprenden durante el entrenamiento y son cruciales para que el modelo capture el significado semántico de los *tokens* [Prince, 2023]. Una de las versiones de GPT-2, por ejemplo, utiliza un vocabulario de 50 257 *tokens* proyectados en un

[7] Una posible traducción al español es «encaje léxico», pero en el área se suelen utilizar los conceptos en inglés directamente por convención. Ver `https://www.hpe.com/lamerica/es/what-is/word-embedding.html`.

espacio de 768 dimensiones, dando un total de 38.6 millones de parámetros solo para esta matriz [Radford *et al.*, 2019, Koti, 2024].

Luego, la matriz de *word embeddings* **X** atraviesa una serie de K capas de *transformers* con **atención enmascarada**, como se observa en la Figura 4. Al principio, cada columna de **X** contiene el significado semántico de cada *token* por sí solo (en el caso de un modelo preentrenado), pero las capas *transformer* se encargan de aplicar el mecanismo de atención con una «máscara» que obliga a cada *token* a poner atención solo en sí mismo y en los *tokens* precedentes, de tal forma que los *embeddings* codifiquen información contextual de forma autorregresiva. Este mecanismo también es denominado *self-attention* en inglés.

Finalmente, los *embeddings* de salida son transformados linealmente al tamaño del vocabulario y una función ***softmax*** se encarga de normalizar los puntajes obtenidos por cada posible *token* a probabilidades, eligiendo aquel con la probabilidad más alta. Durante el entrenamiento, la función de pérdida de entropía cruzada se encarga de maximizar la suma de las log-probabilidades para cada *token* tomando en consideración el *token* real que sigue en cada secuencia de entrenamiento.

2.2.4. Estrategias para muestrear desde un LLM

La tarea de generar salidas novedosas utilizando un LLM basado en *transformers*, una vez entrenado sobre un corpus textual adecuado, se reduce a entregarle como entrada un ***prompt*** (el cual puede ser vacío o, también, un solo *token* que indique el inicio de una secuencia) y dejar que el modelo genere probabilidades para el *token* siguiente, eligiendo el más probable o muestreando desde la distribución generada según diferentes estrategias. Luego, la secuencia concatenada con el *token* elegido por el modelo es entregada como entrada nuevamente, y así sucesivamente hasta algún criterio de término dado, como un largo máximo de secuencia o que el mismo modelo elija un *token* que represente el final de la secuencia [Prince, 2023].

En primer lugar, la elección *greedy* del *token* más probable en cada paso es un proceso determinista y no le da espacio al modelo para ser «creativo». *Beam search* es una técnica que intenta abordar esto al implementar una heurística que evalúa los k mejores candidatos en cada paso antes de elegir el siguiente *token*, pero también es un proceso determinista [Ferruz *et al.*, 2022].

Para fomentar la generación de salidas creativas y novedosas se utilizan técnicas de muestreo aleatorio. La estrategia ***top-k sampling*** elije al azar el siguiente *token* de entre los k *tokens* con las mejores probabilidades. Por otro lado, ***top-p sampling*** elije el siguiente *token* de entre el conjunto más pequeño posible cuyas probabilidades acumuladas sumen un umbral p[8], teniendo la ventaja de que la cantidad de *tokens* posibles en cada paso varía dependiendo

[8]Notar que un valor de $p = 1$ implica elegir de entre todos los *tokens* posibles, por lo que este debe ser menor para que impacte en la elección.

de la certidumbre del modelo [Chornyi, 2024].

También existen estrategias que escalan los *logits*, *i. e.*, los valores de las predicciones crudas, previo a la normalización con una función *softmax*. La **temperatura** T es una de ellas: si $T < 1$, se aumenta la probabilidad de los *tokens* con mejor puntaje, haciendo al modelo más conservador y predecible; si $T > 1$, se aplana la distribución de probabilidad, haciendo al modelo más aleatorio y creativo [Chornyi, 2024]. Por otro lado, las **penalizaciones por repetición** reducen la probabilidad de elegir *tokens* que hayan aparecido recientemente en el texto generado [Siddique, 2023].

2.2.5. Costos computacionales y optimizaciones

La implementación de un LLM basado en *transformers* y el mecanismo de atención es muy costosa. En primer lugar, se requiere de un conjunto de entrenamiento lo suficientemente grande, idealmente en el orden de las miles o millones de secuencias. Luego, la arquitectura misma del entrenamiento de estos modelos requiere paralelizar el proceso: cargar la red neuronal con todos sus parámetros a la memoria y procesar las secuencias de cada *batch* de forma concurrente. Adicionalmente, el tamaño de la ventana de contexto deseada (*i. e.*, el largo de cada secuencia de entrada) tiene una complejidad espaciotemporal cuadrática debido al mecanismo de atención [Duman Keles *et al.*, 2023]. Lo anterior crea la necesidad de acceder a máquinas con una gran capacidad de cómputo, pero los costos asociados generan una barrera de acceso, por lo cual se han desarrollado diferentes enfoques para reducir y optimizar costos.

El primero, y el más directo, es el **ajuste fino**. Un LLM con millones o miles de millones de parámetros es preentrenado sobre un corpus textual masivo y diverso con el objetivo de aprender patrones y generalidades de uno o más lenguajes específicos. Luego, un usuario puede tomar el modelo resultante y volver a entrenarlo sobre un conjunto de secuencias más pequeño y con un contenido más específico, con el objetivo de «ajustar finamente» los pesos del modelo, sesgándolo hacia el contenido del nuevo conjunto de entrenamiento. Si bien este proceso requiere, en principio, la carga en memoria de todos los parámetros del modelo original, los costos asociados disminuyen considerablemente, ya que se puede reutilizar el esfuerzo computacional del preentrenamiento del modelo para adaptarlo a una tarea más específica.

Un tipo especial de ajuste fino para LLMs es el **ajuste de instrucciones**. Esta es una técnica que busca combinar el objetivo de predicción autorregresivo de un LLM con el objetivo de los usuarios de que estos modelos se adhieran a instrucciones específicas, mejorando el control sobre las salidas esperadas. Para lograr esto, se reentrena el modelo con un conjunto de pares `(instrucción, salida)`, donde la `instrucción` se da en lenguaje natural y la `salida` en el lenguaje objetivo, *e. g.*, lenguaje de proteínas o lenguaje natural [Zhang *et al.*, 2024].

En la misma línea, se han propuesto métodos de ajuste fino que buscan hacer más eficiente el proceso a través del desarrollo de algoritmos de entrenamiento que trabajen solo con un subconjunto del total de los parámetros de un modelo extenso. Esto es conocido como «ajuste fino eficiente en parámetros» o **PEFT**, por sus siglas en inglés [Han *et al.*, 2024]. Una técnica PEFT desarrollada específicamente para trabajar con LLMs basados en *transformers* es la «adaptación de bajo rango» o **LoRA**, por sus siglas en inglés. Los autores de LoRA sostienen que, cuando se *adapta* un LLM preentrenado para la ejecución de tareas específicas, el espacio de parámetros que el modelo realmente utiliza es mucho menor que el número total de parámetros disponibles, por lo que es posible «congelar» los pesos del modelo original y, en vez de calcular los gradientes para todos ellos (una matriz $\Delta\mathbf{W}$), el cálculo se realiza sobre dos matrices de bajo rango, **B** y **A**, las cuales se inyectan en cada capa densa de una arquitectura *transformer*, cuyo producto se convierte en la matriz $\Delta\mathbf{W}$. Esto permite entrenar un número drásticamente menor de parámetros en el ajuste fino. Luego, para la inferencia, las matrices **BA** se fusionan explícitamente con los pesos preentrenados del modelo, permitiendo que el paso hacia adelante ocurra sin introducir latencia adicional en comparación con un modelo ajustado sin adaptaciones [Hu *et al.*, 2022].

Por otro lado, existe una técnica conocida como **acumulación de gradientes**, la cual permite simular un *batch size* superior al soportado en un entorno con un tamaño de memoria insuficiente. Procesar secuencias en lote es vital para aprovechar el paralelismo del *hardware* especializado, como GPUs (*Graphics Processing Units*) y TPUs (*Tensor Processing Units*), estabilizar y regularizar el entrenamiento y maximizar tanto la velocidad como la calidad de los LLMs basados en *transfomers* y *self-attention*. Normalmente, cuando un *batch* de secuencias atraviesa toda la red neuronal de un LLM, se calculan los gradientes de cada peso respecto a la función de pérdida para ejecutar su actualización correspondiente. Al utilizar acumulación de gradientes, en cambio, se almacenan y acumulan en una matriz adicional los cambios en los gradientes por una cantidad determinada de pasos, para actualizar los pesos solo posteriormente, lo cual tiene el mismo efecto que si se tratara de un solo *batch* con un tamaño efectivo equivalente al producto entre el *batch size* y la cantidad de pasos de acumulación de gradientes.

Varios otros métodos y algoritmos han sido desarrollados con el objetivo de hacer más eficiente el entrenamiento y el ajuste fino de LLMs. Entre ellos se destaca el uso de precisión mixta [Micikevicius *et al.*, 2017], librerías de optimización como DeepSpeed [Majumder y Proskurin, 2021], puntos de control de gradiente [Chen *et al.*, 2016] y, más recientemente, optimizaciones del mecanismo de atención como Flashattention [Dao, 2023].

2.3. Proteínas

Una **proteína** es una biomolécula básica que tiene un papel fundamental en todos los procesos biológicos, tales como: el metabolismo, el transporte de oxígeno, el desarrollo y el crecimiento de los organismos, incluso en la producción de otras proteínas. Es, a su vez, una

macromolécula cuya estructura primaria corresponde a una o más **secuencias lineales de aminoácidos**.

Un **aminoácido** es una molécula orgánica indispensable para la vida conocida en la Tierra. Se han descubierto más de quinientos tipos de aminoácidos en la naturaleza, incluso en otros cuerpos celestes [Naraoka *et al.*, 2023], pero solo veintidós están codificados en el ADN, los cuales se identifican de forma estandarizada con veintidós letras del abecedario latino [IUPAC, 1968]. Estas moléculas presentan una estructura química común que les permite encadenarse entre sí (ver Figura 5): un átomo central de carbono unido a uno de hidrógeno, un grupo amino y un grupo carboxilo en cada extremo (los cuales se encargan de formar enlaces con otros aminoácidos, denominados enlaces peptídicos), y una cadena lateral (grupo R). Esta última es la parte variable del aminoácido, la cual le otorga distintas propiedades fisicoquímicas (*e. g.*, hidrofobia, polaridad, acidez, etc.).

Las **secuencias de aminoácidos** que se forman al encadenarse estos entre sí se pliegan sobre sí mismas producto de sus propiedades fisicoquímicas, dando lugar a las proteínas (ver Figura 6), las cuales pueden contener desde decenas hasta miles de aminoácidos. Las estructuras tridimensionales finales de las proteínas dependen directamente de las secuencias de aminoácidos subyacentes, y son estas las que definen sus funciones y propiedades.

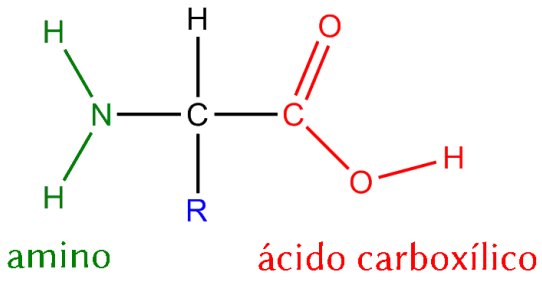


Figura 5: Estructura general de los aminoácidos.
Fuente: Wikimedia Commons.

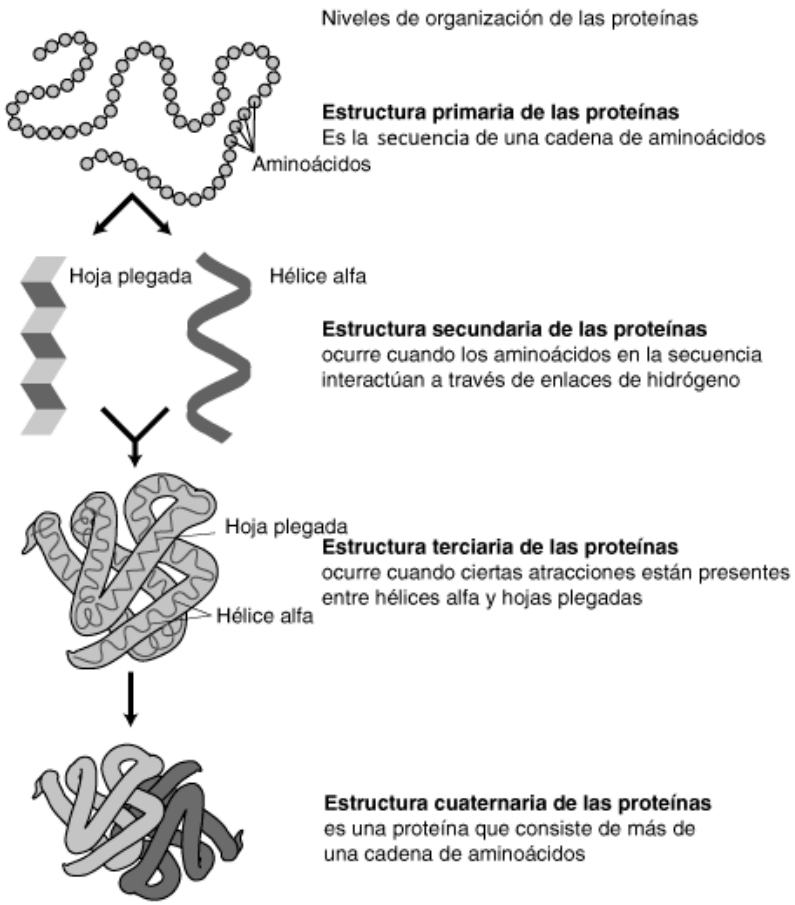


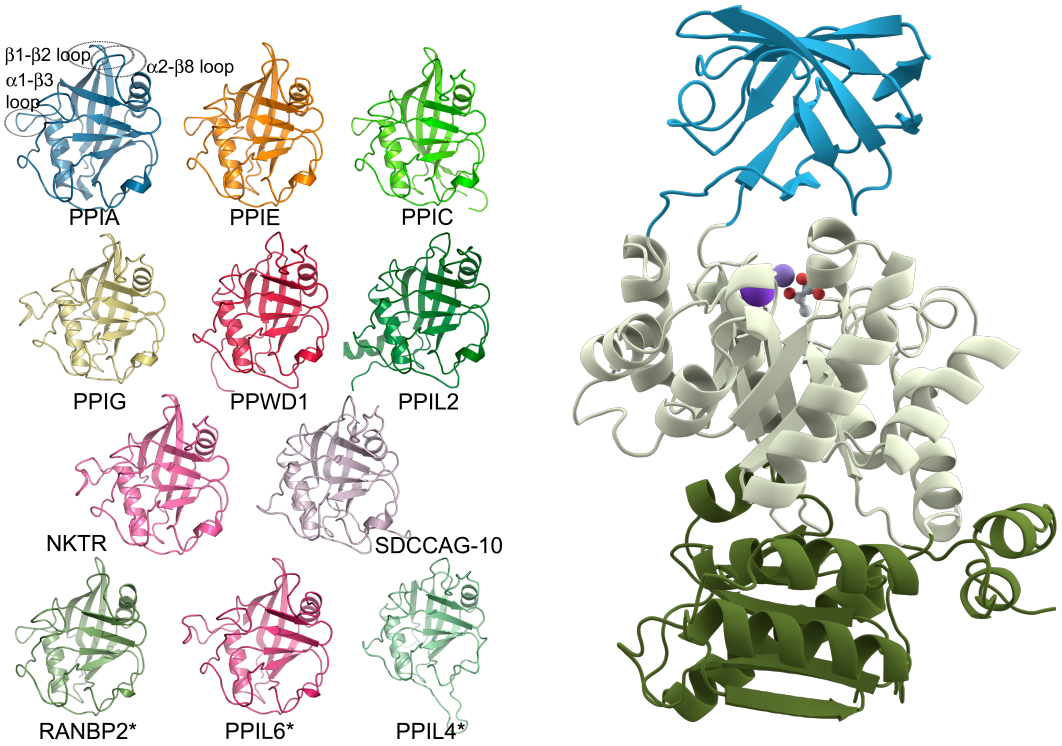
Figura 6: Niveles de organización de las proteínas.
Fuente: Institutos Nacionales de Salud de EE.UU. vía Wikimedia Commons.

2.3.1. Familias de proteínas y dominios conservados

Las proteínas se agrupan según sus relaciones evolutivas mediante las llamadas **familias de proteínas**. En la mayoría de los casos, se relacionan directamente con una familia de genes, ya que un gen es una secuencia de ADN que codifica una proteína. Las proteínas de una misma familia descienden de un mismo ancestro en común y poseen estructuras tridimensionales, funciones y secuencias significativamente similares (ver Figura 7a). Las familias pueden agruparse a su vez en «superfamilias», donde aparecen proteínas que pueden no tener una similitud de secuencia significativa, pero sí de estructura y/o función.

La similitud estructural presente en las proteínas de una misma familia puede explicarse por la existencia de **dominios conservados** (DCs). Un dominio proteico es una parte de la secuencia de aminoácidos de una proteína que puede estabilizarse y plegarse por sí sola, formando una estructura tridimensional compacta, la cual suele corresponderse con una función específica. Los dominios pueden tener un largo de entre 50 y 250 aminoácidos y pueden considerarse unidades modulares, las cuales son utilizadas por la evolución para construir las

proteínas (ver Figura 7b) [Xu y Nussinov, 1998]. Las proteínas de una misma familia pueden contener varios dominios, pero aquellos que se tienen en común corresponden a «motivos» que se han *conservado* a lo largo de la evolución, por lo que se utilizan como un criterio para determinar la pertenencia de una proteína a una familia específica.



(a) Ejemplo de un dominio conservado en una familia de proteínas, la ciclofilina.

(b) Visualización de tres dominios en la proteína piruvato quinasa.

Figura 7: Dominios proteicos y familias de proteínas.
Fuente (a): Extraído de [Davis *et al.*, 2010]. Fuente (b): Wikimedia Commons.

2.3.2. Formato FASTA

El estándar para trabajar computacionalmente con las secuencias de aminoácidos de las proteínas es el **formato FASTA**[9]. Corresponde a un formato que utiliza letras del abecedario latino para almacenar secuencias de nucleótidos o aminoácidos en archivos de texto plano. Un registro FASTA comienza con una descripción de una sola línea, la cual inicia con el carácter «>», seguida de una o más líneas con la secuencia correspondiente.[10] La descripción de una secuencia de aminoácidos usualmente contiene identificadores estandarizados, como el código de acceso en una base de datos pública o la familia de la proteína (ver ejemplo en el Listado 1); sin embargo, esto no es mandatorio.

[9] «FASTA» viene del inglés *fast* (rápido) y *all* (todo). El formato en sí es el legado de un software de alineamiento de secuencias homónimo [Lipman y Pearson, 1985].

[10] Ver `https://zhanggroup.org/FASTA/` para una descripción detallada del formato.

Listado 1: Ejemplo de un registro FASTA de una secuencia de aminoácidos.

```
>A0A009GZY2|PF00083(51...437)|Major facilitator superfamily (MFS) profile domain-containi...
MDLVSRIQNLPIGKFHYTLLWVVGLGWMFDAMDTGIIAFIMTTLVKDWALTPAESGWIVSIGFVGMAIGAVCSGALADRF
GRKTIFAITMAIYSIATALCAFAPDLKWLLIFRFIVGLGLGGQLPVAVTLVSEYIPAHVRGRFIVLLESFWGLGWLVAAL
ISYFIIPKFGWHIAFLMGGLPLIYILVIWKKLPESIPYLINRGRVEEAHALVQKLEAQAGVQIVHHIEVVPVAMRQKVSF
KQLWSGQFARRSLMLWLIWFGIVYSYYGIFTWLPSLLVKQGYDIVKSFEYVLLMILAQLPGYLAAAWFVERLGRKITLAA
FIGFCALSAYFFGQADSVNSIMFWGCLMSFFNLGAWGVLYTYTPEQYPANIRAFGSGWASAMGRLGGILAPIVVTHMMVS
KNGFSHVFMMFTIVLLLVAAVILILGEETQGKRLESIGL
```

2.3.3. Métricas de similitud de secuencias

Desde la bioinformática se han desarrollado varios algoritmos que buscan cuantificar la similitud existente entre dos secuencias de aminoácidos distintas. Métricas para cadenas de caracteres que no consideren el contexto biológico (*e. g.*, la distancia de Levenshtein) no sirven para este efecto, ya que, por ejemplo, podría sustituirse cada aminoácido en una secuencia por otro que tenga propiedades fisicoquímicas similares y obtener así una proteína distinta pero similar a la original, lo cual debería tomarse en consideración al momento de compararlas.

En general, se busca realizar un **alineamiento de secuencias**, consistente en tomar dos o más secuencias de aminoácidos y «alinearlas» de forma que las posiciones homólogas queden frente a frente (ver ejemplo en Figura 8). Al revisar un alineamiento, una métrica básica es el **porcentaje de identidad**, el cual cuenta el número de aminoácidos idénticos en cada región alineada y lo divide por el largo total de cada región.

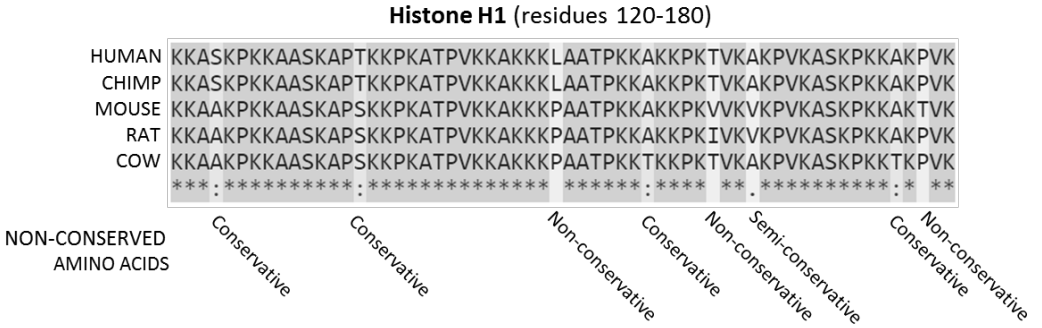


Figura 8: Alineamiento de secuencias de proteínas histonas de diferentes mamíferos. Las columnas destacadas en gris muestran partes conservadas entre todas las secuencias. Fuente: Dr. Thomas Shafee vía Wikimedia Commons.

Existen algoritmos de alineamiento «global» y «local»; en los primeros se busca comparar la longitud total de todas las secuencias, mientras que en los segundos se buscan regiones de máxima similitud dentro de toda la longitud de las secuencias, descartando las partes que no aportan similitud significativa. Para decidir la relevancia de un alineamiento se utilizan matrices como PAM (del inglés *Point Accepted Mutation*) o BLOSUM (del inglés *BLOcks of*

amino acid Substitution Matrix), las cuales contienen puntajes que corresponden a la probabilidad de que un aminoácido sea sustituido por otro basándose en observaciones y/o extrapolaciones de la evolución natural [Henikoff y Henikoff, 1992].

Un algoritmo que encuentra alineamientos locales es el de Smith-Waterman, el cual se basa en programación dinámica y garantiza la optimización de la medida de similitud utilizada (*e. g.*, una matriz de sustitución) [Smith y Waterman, 1981]. Sin embargo, sucesivas heurísticas se han desarrollado a partir de este algoritmo para abordar los alineamientos locales, ya que su complejidad temporal y computacional es cuadrática, por lo que no es útil para ser aplicado en alineamientos con múltiples secuencias en las grandes bases de datos de proteínas actuales.

El algoritmo heurístico basado en Smith-Waterman más utilizado hoy en día es **BLAST** (del inglés *Basic Local Alignment Search Tool*). Este toma una o más secuencias de consulta y las compara contra todas las secuencias de una base de datos objetivo. Por cada secuencia de consulta, el algoritmo ejecuta los siguientes pasos:

1. Fragmenta la secuencia en «palabras» de un tamaño determinado (tres aminoácidos por defecto).
2. Localiza «semillas», *i. e.*, segmentos de secuencias objetivo que coincidan con las palabras del paso anterior.
3. Extiende las semillas en ambas direcciones mientras la puntuación siga aumentando, utilizando una matriz de sustitución y penalizaciones por *gap*[11].
4. Filtra las extensiones cuyo puntaje sea inferior a un umbral predefinido, descartando alineamientos que no sean estadísticamente significativos.

BLAST determina la significancia estadística de un alineamiento local utilizando una matriz de sustitución definida por el usuario, siendo BLOSUM62[12] la matriz por defecto. Para ello, se suman las puntuaciones de la matriz y las penalizaciones por *gaps* en un puntaje bruto S, el cual se normaliza para poder comparar la significancia de distintos alineamientos, obteniendo un puntaje llamado «***bit score***» S'. Finalmente, se obtiene una métrica llamada «***E-Value***» E según la siguiente ecuación:

$$E = 2^{-S'} mn \tag{6}$$

donde m es el largo de la secuencia de consulta y n el largo de las secuencias objetivo. Este valor tiene la interpretación de ser el número de alineamientos con el mismo puntaje que uno podría encontrar por azar en la base de datos objetivo; *i. e.*, si el *e-value* es cercano a

[11] Del inglés para «hueco», espacio introducido en el alineamiento de secuencias que representa una inserción o eliminación de uno o más aminoácidos durante la evolución

[12] Matriz calculada según las sustituciones observadas entre secuencias de proteínas con un máximo de 62 % de identidad [Eddy, 2004].

cero entonces el puntaje del alineamiento no puede ser por azar, mientras que si es cercano a uno o mayor, el puntaje *puede* haberse obtenido por azar.[13] Es el *e-value* el valor utilizado como umbral para filtrar las alineaciones encontradas por BLAST.

2.3.4. *Software* para la identificación de dominios conservados

Identificar los dominios conservados presentes en una o más secuencias de proteínas es una tarea común en bioinformática. Por ello, se han desarrollado varias aplicaciones web que permiten realizarlo de forma eficiente y personalizada a través de interfaces gráficas en el navegador o llamadas a APIs.

Una de las más reconocidas es la base de datos de dominios conservados (CDD, por sus siglas en inglés) y herramientas para el análisis de proteínas del Centro Nacional para Información en Biotecnología de Estados Unidos [Yang *et al.*, 2020]. La batería de servicios públicamente disponibles que se ofrecen incluye el buscador de dominios conservados **Batch CD-Search**, el cual utiliza RPS-BLAST (*Reverse Position-Specific BLAST*), una versión modificada de BLAST que utiliza matrices de puntuación precalculadas sobre una o varias bases de datos de dominios conservados para encontrar alineamientos rápidamente con una o más secuencias de búsqueda [Marchler-Bauer y Bryant, 2004].

2.4. Diseño de proteínas *de novo*

Históricamente, el diseño de proteínas *de novo* ha sido una gran ambición de los biólogos. Si bien el lanzamiento del modelo de IA AlphaFold permitió considerar resuelto el problema del plegamiento de las proteínas, el diseño de proteínas *de novo* es considerado una arista del problema de plegamiento «inverso», donde se busca predecir una secuencia de aminoácidos que se pliegue correctamente a una estructura tridimensional determinada [Richardson y Richardson, 1989]. Existen autores que opinan que las proteínas *de novo* tienen la ventaja de no acarrear un historial evolutivo, por lo que pueden servir como andamios para la construcción de, virtualmente, cualquier función [Ferruz *et al.*, 2022].

En la Introducción (ver página 1) se mencionó que en los primeros intentos de ensamblar proteínas *de novo* nació la bioinformática y en el Capítulo 1, «Definición del problema» (ver página 4), se destacó que existen dos grandes categorías actuales en las cuales se han aplicado modelos de GenAI para producir secuencias de proteínas *de novo*: modelos basados en la estructura y modelos basados en la secuencia de aminoácidos, siendo en esta última categoría donde se puso el foco para el análisis realizado.

[13] Ver `https://www.nlm.nih.gov/ncbi/workshops/2023-08_BLAST_evol/e_value.html` para una explicación más detallada.

Como se mencionó previamente, los LLMs tienen un gran potencial para aprender el «lenguaje» de las proteínas y generar avances importantes en el tratamiento de este problema. Adicionalmente, se mencionó que existen autores que proponen que estos modelos podrían ayudar a explorar partes del espacio de proteínas que no han sido aprovechadas en la naturaleza. Por otro lado, existen autores que hipotetizan que la vida en la Tierra ha tenido suficiente tiempo como para explorar todo el **espacio funcional** de proteínas (que sería apenas una fracción del astronómico número total de posibles combinaciones de aminoácidos), pero que muchas de ellas se pueden haber perdido a lo largo de millones de años debido a factores externos como catástrofes o extinciones masivas, por lo que no se habrían podido conservar hasta el tiempo presente [Dryden *et al.*, 2008].

A día de hoy, se han hecho avances significativos utilizando métodos basados en la estructura deseada, como con aquellos basados en procesos de difusión [Watson *et al.*, 2023]. Sin embargo, los métodos basados en LLMs presentan algunas ventajas destacables por sobre los primeros. En primer lugar, las bases de datos de secuencias de proteínas son mucho más grandes que las de estructuras proteicas y, luego, los PLMs permiten que la generación de proteínas *de novo* se realice prescindiendo de la información estructural, lo cual es especialmente útil cuando esto no se conoce con certeza [Lee *et al.*, 2025]. Por otro lado, existen varias líneas de evidencia a favor de la idea de que la secuencia de aminoácidos de una proteína contiene toda la información necesaria para construir su estructura tridimensional general [Nelson *et al.*, 2019], fundamentando así la importancia del desarrollo de modelos que profundicen el entendimiento del lenguaje de las proteínas.

2.4.1. Modelos de lenguaje de proteínas para el diseño *de novo*

Los LLMs desarrollados para aprender el lenguaje de las proteínas son los denominados **modelos de lenguaje de proteínas** (PLMs, por sus siglas en inglés). Los avances de los últimos años han demostrado que la capacidad de un LLM de entender un determinado lenguaje es directamente proporcional a su tamaño, *i. e.*, a medida que crece el número de parámetros de un modelo de lenguaje, mejor desempeño tiene. Esto constituye una gran ventaja para los PLMs [Toews, 2023].

En el estado del arte se pueden encontrar varios tipos de PLMs, pero solo algunos utilizan arquitecturas específicas para el diseño de secuencias *de novo*. Un estudio reciente clasificó las aplicaciones de los PLMs actuales en cuatro categorías [Lee *et al.*, 2025]:

1. PLMs entrenados sobre bases de datos de proteínas generales. Pueden ser utilizados para generar proteínas de todo tipo, así como también variantes mejoradas de proteínas existentes.

2. PLMs especializados para el diseño de proteínas de familias específicas, a través del entrenamiento sobre conjuntos de datos especializados.

3. PLMs que usan el ajuste fino y el aprendizaje por transferencia para explotar la información aprendida en el espacio de proteínas naturales general y aplicarla en una tarea específica.

4. PLMs multimodales para el diseño de proteínas, los cuales incorporan información estructural o de otro tipo para facilitar el diseño.

En la Figura 9 se muestra un resumen de esta clasificación.

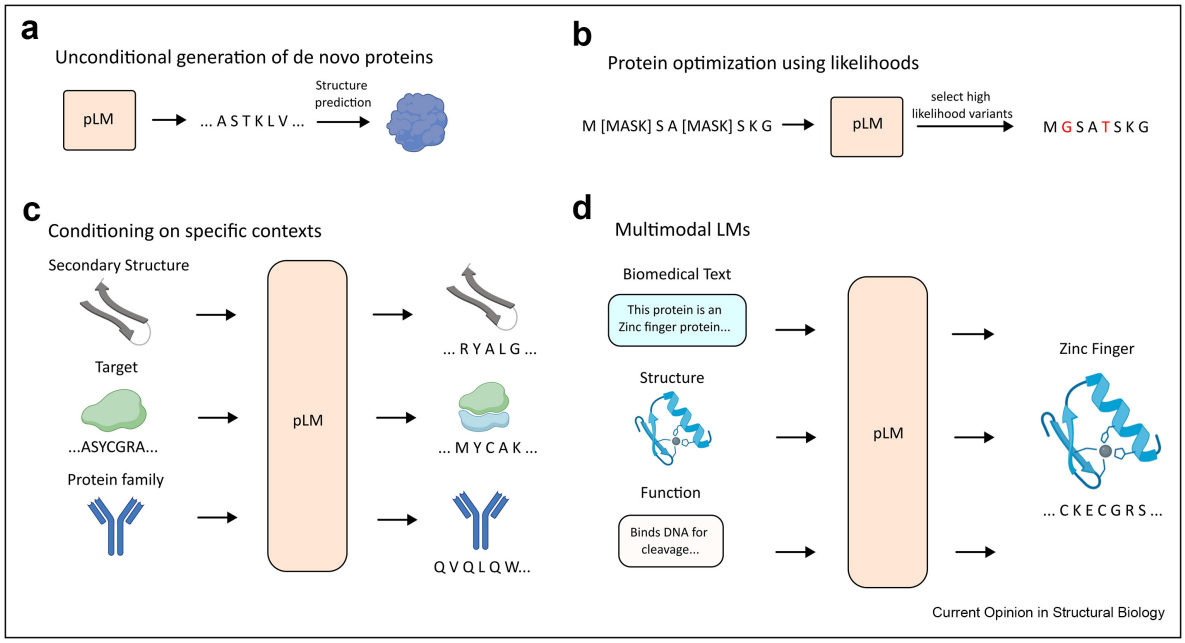


Figura 9: Tipos de PLMs para la generación de secuencias de proteínas *de novo*. (a) Generación de secuencias *de novo* sin condicionar el proceso. (b) Optimización de proteínas generando variantes con alta verosimilitud. (c) Generación de secuencias condicionando sobre contextos específicos, como una estructura objetivo o una familia de proteínas. (d) Generación de secuencias con PLMs multimodales, recibiendo de entrada descripciones en lenguaje natural y/o una estructura deseada. Fuente: Extraída de [Lee *et al.*, 2025].

CAPÍTULO 3
PROPUESTA DE SOLUCIÓN

En este capítulo se presenta la propuesta de solución diseñada para abordar la problemática principal, tal como fue presentada en el Capítulo 1, la cual se relaciona con la falta de trabajos que analicen exhaustivamente modelos de GenAI que producen secuencias de proteínas *de novo*. En particular, y en concordancia con los objetivos definidos en la sección 1.2, **se traza un plan que incluye seleccionar modelos del estado del arte, ejecutar ajuste fino sobre ellos utilizando hiperparámetros óptimos y, finalmente, generar y evaluar secuencias *de novo* generadas por los modelos ajustados**. Lo anterior se realizará siempre tomando en cuenta las limitaciones de tiempo y de costo computacional, así como, también, analizando la capacidad de los modelos para reconstruir dominios conservados en secuencias generadas con un bajo porcentaje de similitud en relación con las secuencias de entrenamiento utilizadas.

Considerando las definiciones conceptuales descritas en el capítulo anterior, en primer lugar, se realizó un estudio de modelos de GenAI *ad hoc*, a partir del cual se decidió limitar el análisis a tres PLMs del estado del arte, en concordancia con el objetivo específico **O1**. Luego, se ejecutaron pruebas de concepto con la implementación de los modelos elegidos, asegurando así un correcto funcionamiento en términos de ajuste fino y de generación de secuencias condicionadas, cumpliendo con parte del objetivo específico **O3**. Finalmente, se diseñó un plan para la evaluación de los modelos, el cual incluyó la elección y preprocesamiento de un conjunto de secuencias de entrenamiento pertenecientes a una familia de proteínas específica, donde se determinaron métricas bioquímicas robustas para la evaluación posterior de los modelos y se desarrolló un algoritmo para su ejecución, todo en función del objetivo específico **O2**.

3.1. Estudio general del estado del arte

Tal como se ha comentado anteriormente, primero se realizó un estudio general del estado del arte respectivo a modelos de GenAI que generan secuencias de proteínas *de novo*. De allí se identificaron las dos categorías mencionadas en el capítulo anterior, *i. e.*, modelos que aprenden de las secuencias de aminoácidos y modelos que aprenden de las estructuras tridimensionales.

3.1.1. Modelos que aprenden de las estructuras de las proteínas: modelos de difusión

Desde la vereda de los modelos que aprenden de las estructuras, se destacan aquellos basados en **procesos de difusión**. Estos modelos, llamados «de difusión», se basan en la idea de

«sistemática y lentamente destruir la estructura en una distribución de datos a través de un proceso iterativo de difusión hacia adelante para, luego, aprender un proceso de difusión inversa que restaura la estructura de los datos, obteniendo un modelo generativo de los datos altamente flexible y manejable»[14] [Sohl-Dickstein *et al.*, 2015]. De esta forma, un modelo de difusión que aprende a generar proteínas *de novo* toma una estructura proteica, la convierte en ruido y, luego, aprende un proceso de difusión para revertir el ruido a la estructura original. En los últimos años, los modelos generativos basados en difusión han demostrado un desempeño superior a modelos con otras arquitecturas previamente dominantes, tales como las redes neuronales adversarias o los *autoencoders* variacionales [Guo *et al.*, 2023].

Algunos modelos de difusión destacados son los siguientes:

- **RFdiffusion**. Es uno de los más populares, ya que ha sido validado ampliamente y es de código abierto, habiendo demostrado su eficacia en el diseño de secuencias con estructuras y funciones deseadas, tales como sitios de unión de proteínas, sitios activos enzimáticos y andamiaje de motivos simétricos para el diseño de proteínas terapéuticas y de unión a metales [Watson *et al.*, 2023].
- **Chroma**. Modelo programable que combina procesos de difusión con redes neuronales gráficas para generar estructuras tridimensionales y secuencias bajo diversas restricciones, obteniendo eficientemente proteínas novedosas, funcionales y de alta fidelidad. Las estructuras de más de 300 proteínas generadas fueron validadas *in vivo*[15] [Ingraham *et al.*, 2023]. Los pesos del modelo están disponibles gratuitamente solo para fines no comerciales.
- **Genie2**. Modelo de código abierto caracterizado por el uso de un *framework* multimotivo, permitiendo la generación de proteínas condicionadas y no condicionadas con múltiples sitios funcionales, sin la necesidad de especificar posiciones ni orientaciones en el espacio tridimensional [Lin *et al.*, 2024].

La principal desventaja de este tipo de modelos radica en que no fueron diseñados para poner atención al lenguaje de las proteínas presente en las secuencias de aminoácidos. Otras desventajas asociadas fueron mencionadas en la sección 2.4.

3.1.2. Modelos que aprenden de las secuencias de las proteínas: PLMs

Como se mencionó anteriormente, los modelos de lenguaje de proteínas (PLMs), *i. e.*, LLMs que aprenden directamente el lenguaje de las proteínas, presentan algunas ventajas por sobre los modelos de difusión al perseguir la tarea de generar secuencias de proteínas *de novo*.

[14] Traducción propia.

[15] Locución latina que, en bioquímica, se refiere a experimentos o estudios realizados dentro de organismos vivos, *e. g.*, bacterias.

Entre ellas se incluyen una mejor capacidad para explorar el espacio funcional de las proteínas, la masificación de las bases de datos de secuencias de proteínas y la hipótesis de que la secuencia de aminoácidos de una proteína basta para construir su estructura tridimensional.

Algunos PLMs destacados son los siguientes:

- **ESMFold**. Gran modelo de lenguaje enfocado en la predicción de las estructuras de proteínas *de novo* basándose solamente en la secuencia de aminoácidos. Se desarrolló bajo la motivación de que otros modelos de predicción de estructuras (*e. g.*, AlphaFold2) requieren la existencia de secuencias homólogas, pero esto, por definición, no ocurre en el caso de las proteínas *de novo* [Lin *et al.*, 2022].

- **ProtGPT2**. Modelo de lenguaje que genera secuencias *de novo* explorando regiones poco conocidas del espacio proteico con rapidez y eficiencia. Su diseño liviano permite su uso en equipos comunes, y ofrece la posibilidad de adaptarse a objetivos específicos mediante el ajuste fino sobre conjuntos de secuencias de interés. Su entrenamiento se realizó únicamente de forma no supervisada, sobre secuencias sin anotaciones [Ferruz *et al.*, 2022].

- **ProGen**. Modelo de lenguaje capaz de generar secuencias de proteínas *de novo* con funciones predecibles a partir de grandes familias de proteínas. El modelo contiene 1.2 miles de millones de parámetros y fue entrenado sobre 280 millones de secuencias de proteínas, aumentando sus capacidades con el uso de etiquetas para la especificación de propiedades [Madani *et al.*, 2023].

- **ProGen2**. Serie de modelos sucesores de ProGen, entrenados sobre más de mil millones de secuencias de proteínas obtenidas desde repositorios de todo tipo. El mayor modelo contiene 6.4 miles de millones de parámetros; los autores quisieron aplicar la hipótesis de que, a mayor escala de un modelo de lenguaje, mejor la captura de la gramática subyacente [Nijkamp *et al.*, 2023].

- **ProtAgents**. Plataforma para la ejecución de variadas tareas relacionadas con proteínas, incluyendo diseño *de novo*, basada en el paradigma de agentes (*i. e.*, sistemas que usan IA para interactuar con un ambiente, utilizar herramientas y completar objetivos autónomamente, a nombre de un usuario). Se caracteriza por el uso de un LLM que orquesta a varios agentes para ejecutar las tareas, donde cada uno actúa como un experto en diversos dominios [Ghafarollahi y Buehler, 2024].

- **ProtLLM**. Modelo de lenguaje *cross-modal* (*i. e.*, que utiliza un tipo de entrada para predecir una salida de otro tipo) caracterizado por el manejo de entradas complejas, mezclando lenguaje natural con lenguaje de proteínas. Para ello, proponen un vocabulario especializado de proteínas que denominan «proteína-como-palabra» y un *dataset* propio que incluye datos estructurados (*e. g.*, anotaciones de proteínas) y no estructurados (*e. g.*, artículos de investigación) [Zhuo *et al.*, 2024].

- **ProLLaMA**. Modelo innovador que combina el lenguaje natural con el lenguaje de las proteínas, permitiendo que un sistema interprete instrucciones humanas para generar o analizar secuencias proteicas. Se basa en adaptar un LLM mediante un entrenamiento progresivo, lo cual preserva su conocimiento lingüístico, a la vez que le otorga comprensión del lenguaje biológico. Esto le permite generar respuestas precisas a instrucciones dadas en lenguaje natural [Lv *et al.*, 2024].

Dadas las ventajas presentadas por los PLMs, las necesidades de la empresa y los recursos disponibles, **se decidió elegir tres PLMs del estado del arte para la ejecución del análisis**. Adicionalmente, comparar los resultados de un PLM con los de un modelo de difusión hubiese requerido elegir al menos dos modelos de cada tipo para que el análisis hubiera sido más robusto, y esto se descartó también debido a la mayor accesibilidad presentada por los PLMs del estado del arte en comparación.

3.2. Selección de PLMs del estado del arte

Para seleccionar los tres PLMs sobre los cuales ejecutar el análisis requerido de entre los siete modelos presentados en la sección anterior, se revisó a cabalidad cada uno. A partir de esta revisión, se descartaron modelos siguiendo los criterios que se presentan a continuación.

El primer criterio consistió en definir cuáles modelos realmente podían utilizarse para generar secuencias de proteínas *de novo* condicionadas mediante algún procedimiento de ajuste fino. Con ello, se descartó ESMFold, ya que no genera proteínas *de novo*, si no que busca sus estructuras tridimensionales. También se descartó ProtLLM, por tampoco estar diseñado para ejecutar la tarea de interés.

En segundo lugar, se utilizó un criterio relacionado con la disponibilidad o accesibilidad de los modelos restantes. En este caso, se descartó ProGen, ya que en su repositorio oficial dejaron de existir los pesos al modelo original y solo se encuentran disponibles los enlaces a los pesos de su serie sucesora, ProGen2.[16] Como caso especial, también se descartó ProtAgents debido a que no utiliza un PLM para el diseño de proteínas *de novo*, si no que realiza una llamada al modelo de difusión Chroma.

Con ello, los tres modelos restantes fueron **ProtGPT2**, **ProLLaMA** y **ProGen2**. Estos se caracterizan por poseer una buena documentación para la ejecución de ajuste fino y la generación de secuencias (con excepción de ProGen2, ver subsección 3.2.3). A continuación, se presentan en detalle los tres modelos seleccionados, sus características principales y los hiperparámetros utilizados por sus autores.

[16] Ver repositorio en `https://github.com/salesforce/progen`.

3.2.1. ProtGPT2

Modelo basado en **GPT2-large** capaz de generar secuencias de proteínas *de novo* con un alto rendimiento que exploran áreas «oscuras» del espacio proteico (*i. e.*, proteínas cuyas estructuras no son accesibles mediante métodos experimentales o modelamiento por homología) al expandir las superfamilias naturales. Sus autores destacan la liviandad del modelo, pudiendo «ser utilizado para generar secuencias en estaciones de trabajo estándar en cuestión de segundos, o ser ajustado finamente sobre conjuntos de interés de un usuario para aumentar familias de proteínas específicas» [Ferruz *et al.*, 2022].[17] El modelo tendría el potencial de permitir la detección de proteínas similares a las naturales que podrían mejorar, ajustar o alterar funciones bioquímicas específicas.

ProtGPT2 cuenta con **738 millones de parámetros**, repartidos en una red neuronal profunda de 36 capas, utilizando 1280 dimensiones de *embeddings*, al igual que su modelo base [Radford *et al.*, 2019]. Para codificar el vocabulario, los autores utilizaron el algoritmo conocido como ***byte-pair encoding***, el cual se caracteriza por encontrar las raíces de las palabras más frecuentemente utilizadas, asegurando un mejor desempeño que la codificación *one-hot*; el resultado final fue **un vocabulario con 50256 *tokens*** correspondientes a las combinaciones de aminoácidos más comunes en el espacio proteico, con un tamaño **promedio de 4 aminoácidos por *token***.

En términos de arquitectura y entrenamiento, es el modelo más sencillo de entre los tres elegidos. El entrenamiento base del modelo se realizó de forma no supervisada sobre el conjunto de secuencias de proteínas **Uniref50**[18] (versión `2021_04`), sin anotaciones, utilizando **~45 millones para entrenamiento** y **~5 millones para validación** (*i. e.*, una proporción 90/10). Los autores configuraron una ventana de contexto de 512 *tokens*, equivalente a 2048 aminoácidos en promedio. Luego, para ejecutar el entrenamiento, se reiniciaron los pesos del modelo base y fueron optimizados utilizando **Adam** y una tasa de aprendizaje de 0.001. Finalmente, se utilizó un *batch size* de 8 y 128 GPUs NVIDIA A100, totalizando un *batch size* efectivo de 1024 secuencias y completando el entrenamiento en 4 días.

Dado que ProtGPT2 fue entrenado de forma no supervisada, **solo puede guiarse la generación de secuencias a través del ajuste fino**. Más aún, los autores evaluaron las secuencias generadas por el modelo solo en términos de sus propiedades estructurales y bioquímicas, mostrando que estas repetían la frecuencia relativa de aparición de estructuras secundarias y terciarias de conjuntos de proteínas naturales. No se realizó un análisis respecto a la presencia de dominios conservados.

[17] El modelo permanece a libre disposición a través del repositorio HuggingFace en `https://huggingface.co/nferruz/ProtGPT2`.

[18] Ver `https://www.uniprot.org/uniref` para una descripción de los conjuntos de secuencias Uniref.

Para la ejecución del ajuste fino, se requiere preprocesar las secuencias para que cumplan con el formato mostrado en el Listado 2. Como se observa, se reemplazan los encabezados FASTA por la cadena <|endoftext|> y, si es que las secuencias no vienen con saltos de línea cada 60 caracteres, deben agregarse.

Listado 2: Ejemplo de dos secuencias preprocesadas para el ajuste fino de ProtGPT2.

```
<|endoftext|>
MKDIDTLISNNALWSKMLVEEDPGFFEKLAQAQKPRFLWIGCSDSRVPAERLTGLEPGEL
FVHRNVANLVIHTDLNCLSVVQYAVDVLEVEHIIICGHYGCGGVQAAVENPELGLINNWL
LHIRDIWFKHSSLLGEMPQERRLDTLCELNVMEQVYNLGHSTIMQSAWKRGQKVTIHGWA
YGIHDGLLRDLDVTATNRETLEQRYRHGISNLKLKHANHK
<|endoftext|>
MDGSGDFYDPYTTTSTALSATNPRPPSSSDPSTQQSPNDDDDFILFPSPPHNHTTSSSSC
DHSPPNAKSSSNPHLSPSSSSNTGASSNKISKRRKNTEAARRYRQRKVDRVTELEEALAA
VQKERDELKFRLVRCEAEVEVLRRVGMGK
```

Para generar secuencias, los autores probaron distintos parámetros de inferencia. Obtuvieron los mejores resultados utilizando muestreo *top-k* con $k = 950$ y aplicando una penalización por repetición de 1.2. Utilizaron el valor 1 por defecto para temperatura y muestreo *top-p*.

3.2.2. ProLLaMA

Framework y modelo basado en **LLaMA2** con la particularidad de combinar el lenguaje natural con el lenguaje de las proteínas. Los autores argumentan que, para que un PLM sea realmente útil, este debe poder recibir instrucciones en lenguaje natural respecto a las salidas deseadas, exponiendo que «el lenguaje natural es completo para tareas de procesamiento del mismo, mientras que el lenguaje de las proteínas no es completo para tareas de procesamiento del lenguaje de las proteínas. [...] las instrucciones y salidas no pueden ser completamente representadas en dicho lenguaje»[19] [Lv *et al.*, 2024]. Por ello, los autores desarrollaron un *framework* que permitiría convertir cualquier LLM en un PLM con manejo de ambos lenguajes de forma eficiente; luego, aplicaron dicho *framework* sobre el modelo LLaMA2, obteniendo así ProLLaMA.

El *framework* implementado divide el entrenamiento en dos fases, utilizando **LoRA** en ambas:

1. Se escoge un LLM autorregresivo y se ejecuta un proceso de **aprendizaje continuo** sobre él, entregándole un gran conjunto de secuencias de proteínas de forma no supervisada. Una vez terminado el entrenamiento, el LLM se convierte en un PLM con conocimiento del lenguaje natural.

[19] Traducción propia.

2. Se ejecuta **ajuste de instrucciones** sobre el PLM de la fase 1, con el objetivo de controlar las salidas generadas. De esta forma, se obtiene un PLM capaz de responder a instrucciones dadas en lenguaje natural.

En la implementación desarrollada por los autores se eligió el modelo LLaMA2, en su versión de **6.9 mil millones de parámetros** [Touvron *et al.*, 2023], para la primera fase. El aprendizaje continuo se realizó sobre **69 millones de secuencias** de la base de datos **UniRef50**, cada una de las cuales fue preprocesada con la adición de prefijos y sufijos específicos para ayudar al modelo a distinguir el nuevo lenguaje de proteínas de su conocimiento previo.

Se utilizó **LoRA** para congelar gran parte de los parámetros del modelo, previniendo el olvido catastrófico, mejorando la escalabilidad y reduciendo sustancialmente el costo computacional. El resultado de la primera fase fue el PLM **ProLLaMA_Stage_1**[20], el cual conserva conocimiento del lenguaje natural gracias a que solo se entrenó la fracción de las matrices de bajo rango definidas con LoRA. En la Figura 10 se observa el detalle de la arquitectura del modelo; los autores decidieron incluir las capas «*Embed*» y «*Generation Head*» en el entrenamiento bajo la premisa de que un *token* podría tener diferentes significados en secuencias de proteínas y de lenguaje natural, por lo que un mismo *token* requeriría distintos *embeddings*.

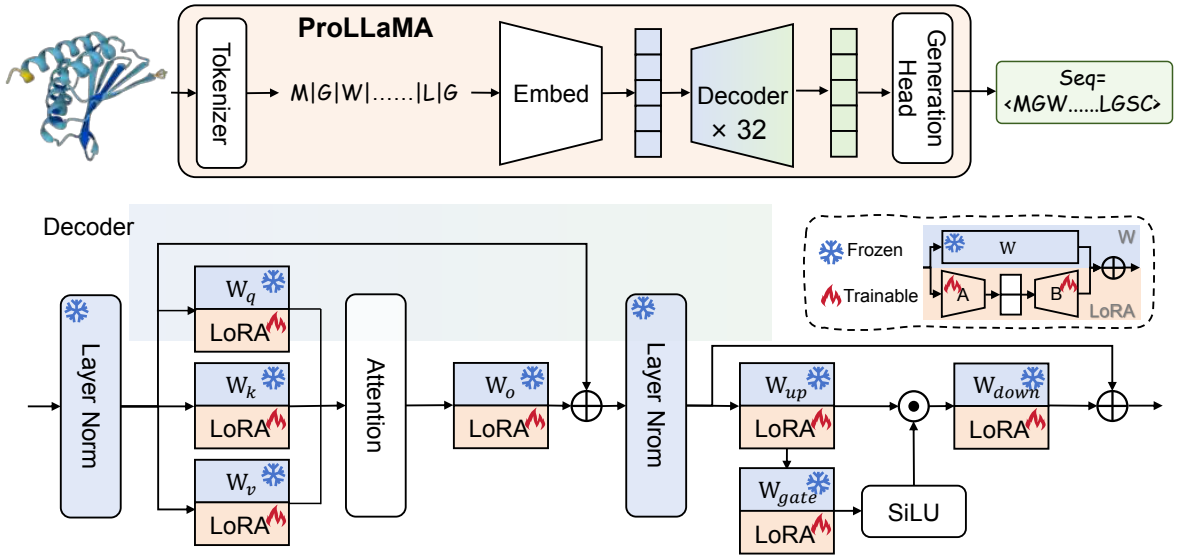


Figura 10: Arquitectura general de ProLLaMA.
Se muestra el detalle de la aplicación de LoRA en algunos de los pesos de las 32 capas *decoder*. Solo ellos se encargan de aprender del lenguaje de las proteínas, el resto son congelados durante el entrenamiento. Fuente: Extraída de [Lv *et al.*, 2024].

Luego, la segunda fase fue ejecutada, siguiendo el formato de ajuste de instrucciones de **Stanford Alpaca** para enseñarle al modelo a responder a instrucciones en lenguaje natu-

[20] El modelo permanece a libre disposición a través del repositorio HuggingFace en `https://huggingface.co/GreatCaptainNemo/ProLLaMA_Stage_1`.

ral [Taori *et al.*, 2023]. Esto se logró adaptando el objetivo del modelamiento causal del lenguaje, la entropía cruzada, haciendo que la predicción del *token* siguiente dependa de una instrucción u, además de los *tokens* anteriores (*i. e.*, la entrada) y los parámetros mismos del modelo. De esta forma, la instrucción en sí solo necesita ser entendida por el modelo, como una etiqueta, ya que no está involucrada en la generación autorregresiva en sí.

En particular, se instruyó al modelo a determinar la superfamilia de una secuencia dada y a generar secuencias *de novo* dada una superfamilia. Esto dio como resultado el modelo final **ProLLaMA**. En el Listado 3 se observa un ejemplo de un conjunto de instrucciones en formato JSON utilizado para el ajuste de instrucciones del modelo, donde se destaca que cada ejemplo puede ser una instrucción distinta, pudiendo utilizar lenguaje de proteínas tanto en la entrada como en la salida.[21]

Listado 3: Ejemplo de conjunto de instrucciones en formato JSON para el ajuste de ProLLaMA.

```
[
    {
        "instruction": "[Generate by superfamily]",
        "input": "Superfamily=<Gnk-homologous domain superfamily>",
        "output": "Seq=<MDAIIWHEFCQIQYSYRMFFSMMVYNGKYPDSNDQE>"
    },
    {
        "instruction": "[Determine superfamily]",
        "input": "Seq=<MRKGLEAWFTLLIESVWGKRRIMEVYLNVAETG>",
        "output": "Superfamily=<Lysozyme-like domain superfamily>"
    },
    ...
]
```

De entre los dos modelos generados en ambas fases, **ProLLaMA_Stage_1** entrega la posibilidad directa de ser ajustado finamente mediante el ajuste de instrucciones. Solo basta generar un conjunto de instrucciones personalizado siguiendo el formato de Stanford Alpaca y utilizar los *scripts* correspondientes a la segunda fase para obtener un modelo que responda a instrucciones de interés, *e. g.*, generar secuencias *de novo* pertenecientes a determinadas familias de proteínas.

La configuración para el ajuste de instrucciones utilizada por los autores consistió en un rango LoRA de 64 con 2 *epochs*. Al cargar el modelo con el *script* utilizado, del total de 6.9 mil millones de parámetros, **solo 160 millones** (equivalente a un 2.32 %) **son entrenables con LoRA**. Se utilizó el algoritmo de optimización **AdamW** y una tasa de aprendizaje programada con un *annealing* cosenoidal con ***warmup steps*** y un valor máximo de 5×10^{-5}. Se fijó el largo máximo de secuencia en 256 y se utilizó un batch size efectivo de 4608 secuencias.[22] El conjunto de instrucciones se construyó intersectando secuencias de **UniRef50** y descrip-

[21] Las secuencias del ejemplo no son reales.

[22] Otros detalles se pueden ver directamente en el *script* disponible en `https://github.com/PKU-YuanGroup/ProLLaMA/blob/main/scripts/run_it.sh`.

ciones textuales de las proteínas en **InterPro**. Para la generación de secuencias se utilizaron los parámetros de muestreo fijados en el *script* de inferencia: temperatura 0.2, top-k con $k = 40$, top-p con $p = 0{,}9$ y penalización por repeticiones 1.2.

3.2.3. ProGen2

Serie de modelos de código abierto que utilizan la arquitectura *Conditional Tranformer Language Model*, la cual permite condicionar la generación de secuencias con la introducción de *tokens* de control especiales de forma autorregresiva [Nijkamp *et al.*, 2023, Keskar *et al.*, 2019]. Los autores afirman que sus modelos «muestran un rendimiento de vanguardia al capturar la distribución de secuencias evolutivas observadas, generar nuevas secuencias viables y predecir la aptitud de las proteínas sin ajuste fino adicional». Existe en sus siguientes versiones:

- **ProGen2-small**: 151 millones de parámetros.
- **ProGen2-medium**: 764 millones de parámetros.
- **ProGen2-OAS**: 764 millones de parámetros, entrenada solo sobre secuencias de anticuerpos naturales.
- **progen2-large**: 2.7 mil millones de parámetros.
- **progen2-xlarge**: 6.4 mil millones de parámetros

Los modelos fueron preentrenados de forma no supervisada sobre un corpus masivo de **mil millones de secuencias** de proteínas provenientes de fuentes genómicas y metagenómicas (**UniRef90** y **BFD30**[23]). Al evaluar el desempeño de los modelos con un conjunto de prueba independiente, los autores descubrieron que los modelos más grandes obtenían perplejidades «sustancialmente más bajas»; sin embargo, hicieron hincapié en que una perplejidad baja solo refleja una buena capacidad del modelo para capturar la distribución de los datos de entrenamiento y no necesariamente una medida de aptitud molecular respecto a las secuencias de proteínas que generan.

Respecto al ajuste fino, los autores afirmaron haber demostrado que, al aplicarlo sobre sus modelos, permitieron que estos redujeran el espacio a explorar para la generación de secuencias de proteínas pertenecientes a familias específicas. Al generar secuencias, los autores señalaron que «los parámetros de muestreo están fuertemente correlacionados con la novedad de una secuencia», *i. e.*, una mayor temperatura o valor p de *top-p sampling*. Así también, parámetros de muestreo más restrictivos típicamente dan como resultado estructuras más parecidas a las de proteínas naturales.

[23] Ver `https://bfd.mmseqs.com/` para una descripción de la base de datos BFD30.

Los pesos de cada versión del modelo original están disponibles en un repositorio público.[24] Sin embargo, a la fecha, este tiene una documentación muy vaga, no se explica cómo ejecutar ajuste fino sobre él e, inicialmente, los enlaces a los pesos de cada modelo se encontraban caídos, por lo que se eligió experimentar con una versión ligeramente distinta desarrollada por [Hrbáň y Hoksza, 2024]. En ese trabajo, **los autores modificaron el modelo original para que pudiera condicionarse la generación con *tokens* especiales**, dejando libre el acceso a los pesos de su versión de ProGen2 y todos los *scripts* necesarios para su ejecución y ajuste fino.[25] En particular, el objetivo fue ejecutar ajuste fino sobre ProGen2 para que este pudiera generar secuencias pertenecientes a familias de proteínas seleccionadas. Para ello, seleccionaron secuencias de siete familias distintas desde la base de datos **Pfam**[26], promediándose **12888 secuencias** por cada una.

El preprocesamiento de las secuencias de proteínas para el ajuste fino consiste en concatenar cada una con los caracteres `1` y `2`, al final y al inicio respectivamente, los cuales actúan como delimitadores; luego, a la cadena resultante se le concatena al inicio un ***token* de control**, *i. e.*, una etiqueta en la forma `<|etiqueta|>`, la cual indica la familia a la que pertenece (o cualquier otra propiedad de interés). Finalmente, se ejecuta el entrenamiento del modelo siguiendo el objetivo de modelamiento autorregresivo con las secuencias preprocesadas, donde las etiquetas ayudan a condicionar el entrenamiento como en el aprendizaje supervisado. Los autores desarrollaron un *script* Python que permite procesar uno o más archivos FASTA con secuencias de proteínas, utilizando los nombres de cada archivo como las etiquetas a concatenar; el resultado es un archivo de texto con el formato mostrado en el Listado 4.

Listado 4: Ejemplo de secuencias de proteínas preprocesadas para ProGen2.

```
<|etiqueta|>1MPQTRKLILDVLKPHQPDALDFSLALADLGVDYRVNLTVTEVDKKTESTIV2
<|etiqueta|>1MAHFQNHYASPAGSRTDEYGNPIRPAD2
<|etiqueta|>1MKSKPSLPLAYRFAVTSRVLAAVIGGYLMASLASICLALWL2
```

Durante la fase de experimentación previa (ver subsección 3.2.5), se observó que el tamaño máximo de modelo ejecutable en la máquina utilizada era **ProGen2-medium**, el cual se utilizó para los experimentos posteriores. Su arquitectura consiste en **764 millones de parámetros** repartidos en 27 capas de neuronas y una ventana de contexto fijada en 1024 *tokens*. El **vocabulario contiene 30 *tokens***: 25 letras para 25 aminoácidos (incluyendo `X` para representar uno desconocido), los delimitadores `1` y `2`, y los *tokens* especiales `<|pad|>` (*token* de relleno), `<|bos|>` (inicio de secuencia) y `<|eos|>` (fin de secuencia). Los autores desarrollaron un *script* de ajuste fino que identifica los ***tokens* de control** en los conjuntos de entrenamiento y validación y genera nuevos *embeddings* para agregarlos al vocabulario original del modelo.

Los hiperparámetros utilizados por los autores en el ajuste fino fueron: un *batch size* efectivo de 64 (utilizando pasos de acumulación de gradiente), 5 *epochs*, tasa de aprendizaje 1×10^{-4}

[24] Ver repositorio en https://github.com/salesforce/progen/tree/main/progen2.
[25] Ver repositorio en https://github.com/hugohrban/ProGen2-finetuning.
[26] Acceso a la base de datos en https://www.ebi.ac.uk/interpro/entry/pfam/#table.

y 400 *warmup steps*; afirmando que los primeros tres fueron los hiperparámetros con mayor influencia en el desempeño final del modelo [Hrbáň y Hoksza, 2024]. Además, utilizaron el optimizador **Adam** con un decaimiento cosenoidal de la tasa de aprendizaje y un crecimiento lineal de los *warmup steps*. Para la generación autorregresiva de secuencias, los autores obtuvieron los mejores resultados utilizando muestreo *top-k* con $k = 15$ y dejando los otros parámetros (*e. g.*, temperatura) en sus valores por defecto.

3.2.4. Resumen de los modelos elegidos

En la Tabla 1 se muestra información que resume y compara las principales características de los PLMs elegidos.

Tabla 1: Cuadro comparativo de las principales características de los PLMs elegidos.

Propiedad/Modelo	**ProtGPT2**	**ProGen2-medium**	**ProLLaMA**
N° parámetros	738M	764M	6900M
Modelo base	GPT2-large	N/A	LLaMA2
Tamaño vocabulario	50k *tokens*	30 *tokens* + *control tags*	32k *tokens*
Aminoácidos por token	4 en promedio	1	No hay datos.
Datos preentrenamiento	50M secuencias	1000M secuencias	69M secuencias
Ajuste fino	No supervisado	Con *control tags*	Con ajuste de instrucciones

3.2.5. Pruebas de concepto con los PLMs seleccionados

A partir del estudio en profundidad de cada modelo, desde AQUIT se entregó una serie de tres conjuntos de secuencias de proteínas con el objetivo de ejecutar los primeros experimentos con los modelos seleccionados. En la Tabla 2 se muestran los tamaños de cada conjunto, el valor medio de los largos de sus secuencias, junto a la desviación estándar de cada valor, y el largo máximo de secuencia presente.

Las secuencias de cada conjunto fueron curadas manualmente y son principalmente monodominio. De entre ellas, solo la proteína quinasa A corresponde a una familia de proteínas, mientras que las otras dos se definen más bien como superfamilias de motivos estructurales que se pueden encontrar en proteínas pertenecientes a familias distintas. Se observa que todos los conjuntos presentan una alta desviación estándar en el largo de las secuencias y, además, su tamaño es bastante más pequeño que el de los conjuntos que fueron utilizados

Tabla 2: Conjuntos de secuencias utilizados para la ejecución de pruebas de concepto.

Familia o motivo estructural	N° de secuencias	Largo promedio	Largo máximo
Dedo de zinc (**ZF**)	139	1221.1 ± 1460.6	7555
Proteína quinasa A (**PKA**)	144	597.2 ± 368.8	2828
Zipper de leucina (**LZ**)	324	458.0 ± 307.2	2917

para preentrenar los modelos seleccionados (del orden de las millones de secuencias). Esto último se debe a que, en AQUIT, las proteínas objetivo son considerablemente menos comunes, por lo que la cantidad de secuencias disponibles para ejecutar el ajuste fino se reduce bastante. Generar secuencias *de novo* pertenecientes a familias de proteínas poco representadas requiere un cuidadoso ajuste fino, por ello **se hipotetizó que, para que los modelos tuvieran un buen desempeño, sería necesario optimizar sus hiperparámetros para obtener el mayor aprendizaje posible desde conjuntos de este tamaño**.

Así, como parte de la experimentación previa con los *scripts* de los modelos, se ejecutó ajuste fino y generación de secuencias sobre cada uno, utilizando sus hiperparámetros por defecto con cada conjunto de la Tabla 2. En principio, se experimentó en **el clúster *cloud* Leftraru Epu del NLHPC-Chile**, el cual contaba con solo una partición con dos GPUs NVIDIA, las cuales son necesarias para la ejecución de los modelos en paralelo, pero la memoria GPU era insuficiente (16 GB cada una). En consecuencia, se optó por utilizar **una partición de Google Colab Pro con una GPU NVIDIA A100 de 40 GB** y se tuvo como objetivo adicional determinar las condiciones bajo las cuales los modelos podían ejecutarse correctamente en el entorno de *hardware* disponible. A partir de esto, se determinaron algunas limitaciones, tales como los los tamaños máximos de cada *batch* y, en el caso de ProGen2, el tamaño máximo de modelo ejecutable. En particular, se estimó que una GPU NVIDIA A100 de 80 GB hubiera sido suficiente para ejecutar la versión más grande de ProGen2, pero no se tuvo acceso a una máquina así.

Luego, se ejecutó **Batch CD-Search** sobre las secuencias generadas por cada modelo, para cada conjunto de proteínas, para revisar si contenían los dominios conservados esperados. Los resultados fueron, como se esperaba, bastante deficientes. En la Tabla 3 se muestran los porcentajes del total de secuencias generadas por cada modelo, para cada conjunto de entrenamiento, que contenían los dominios conservados esperados.

Tabla 3: Porcentajes de recuperación de dominios conservados en las secuencias generadas.

Modelo	Recuperación DCs ZF	Recuperación DCs PKA	Recuperación DCs LZ
ProGen2	10.6 %	4.2 %	40.5 %
ProLLaMA	0.0 %	0.0 %	3.2 %
ProtGPT2	1.3 %	2.2 %	0.0 %

3.3. Diseño experimental

Como se mencionó en la sección anterior, la ejecución *out-of-the-box* del ajuste fino de los PLMs seleccionados no sirve, ya que se requiere optimizar sus hiperparámetros para las características particulares del conjunto de secuencias a utilizar, a la vez que se toman en cuenta las restricciones de tiempo y capacidad de cómputo existentes. La mejor arquitectura a la que se tuvo acceso fue a la de **la partición A100 de Google Colab Pro**, la cual se utilizó para ejecutar todos los procesos de cómputo. Esta contaba con las siguientes especificaciones:

- **RAM del sistema**: 83.5 GB
- **GPU**: NVIDIA A100 40 GB VRAM
- **Almacenamiento**: 235.7 GB

Una vez definido el entorno, se procedió a ejecutar las pruebas de concepto de los modelos mencionadas en la sección anterior. Una de las observaciones más relevantes hechas en ese proceso fue que las secuencias con un largo mayor a las ventanas de contexto definidas eran truncadas por los modelos antes de ser procesadas, por lo que las secuencias de entrenamiento deben tener un largo igual o menor. Así, se advirtieron esta y otras condiciones que debía cumplir el conjunto de entrenamiento a utilizar, además de la necesidad de optimizar los hiperparámetros para cada modelo.

3.3.1. Construcción de conjuntos de secuencias para entrenamiento y validación

Para comparar efectivamente el desempeño de los modelos entre sí, así como con distintos hiperparámetros, se decidió ejecutar el ajuste fino de cada uno sobre un mismo conjunto de secuencias, para posteriormente generar secuencias *de novo* y analizar su calidad. Dado lo observado en las pruebas de concepto, se tomaron las siguientes consideraciones respecto al conjunto de secuencias de entrenamiento a elegir:

- No debe contener más de mil secuencias, ya que desde AQUIT no utilizarán un número mayor de secuencias para sus experimentos.
- Ninguna secuencia debe tener un largo mayor a 1024 aminoácidos, ya que ese es el tamaño de la ventana de contexto definida en los modelos.
- Todas las secuencias deben comenzar con el aminoácido `M` (metionina), que es el primero en ser producido durante la traducción de ARN mensajero [Guedes *et al.*, 2011].

La ventana de contexto podría ser mayor pero, como se mencionó en la subsección 2.2.5, la ocupación de memoria aumenta de forma cuadrática con el largo de la secuencia. Además, cuando las secuencias de un largo mayor son truncadas por los modelos, estas pierden significado biológico.

Por otro lado, la restricción del aminoácido de inicio tiene relación con que fue imposible lograr que el modelo ProGen2 ajustado generara secuencias con un *prompt* vacío (*i. e.*, generación *de novo*). Entonces, se identificó el siguiente problema: si el modelo aprende que las secuencias pueden comenzar con distintos aminoácidos durante el entrenamiento, pero, al generar secuencias, se debe entregar sí o sí uno de inicio como parte del *prompt*, se estaría acotando el espacio de posibles secuencias a generar a aquellas que inician con el aminoácido entregado. Para subsanar esto, se consideró filtrar del conjunto de entrenamiento todas las secuencias que no comenzaran con el aminoácido `M` con el objetivo de que los ajustes finos de los tres modelos se hicieran bajo las mismas condiciones, es decir, que aprendieran una gramática probabilística donde las secuencias siempre inician con `M`. Esta decisión se sostiene, también, sobre el hecho de que la mayoría de las secuencias de aminoácidos en las bases de datos de proteínas comienzan con dicho aminoácido.

Una vez definidos los criterios de selección, se propuso elegir una familia de proteínas numerosa (con decenas o centenas de miles de proteínas secuenciadas) presente en alguna base de datos pública (*e. g.*, InterPro [Blum *et al.*, 2024]). Luego, filtrar todas aquellas secuencias que cumplan con los criterios mencionados y seleccionar al azar, sin repetición, mil de ellas. **Del total, el 95 % (*i. e.*, 950 secuencias) se destinarán para el entrenamiento de los modelos, y el 5 % restante (*i. e.*, 50 secuencias) para validación**. La razón detrás de la proporción elegida tiene relación con que la evaluación final de cada modelo ajustado se realizará con el análisis posterior de las secuencias que generan, por lo que las secuencias de validación sólo servirán para llevar un registro del desempeño aproximado del modelo al final de cada *epoch*; además, de por sí se trata de un conjunto bastante reducido, por lo que quitarle demasiadas secuencias al entrenamiento sería contraproducente.

Finalmente, se utilizó la plataforma en línea InterPro para encontrar una familia de proteínas numerosa y, dentro de lo posible, bien documentada. Se decidió por la **familia de transportadores de glucosa**, la cual contaba con 339 mil secuencias documentadas, de entre las cuales **se extrajeron 1000 secuencias al azar que cumplieran con los criterios estipulados anteriormente**. Además, en la misma plataforma aparecen al menos 86 estructuras tridimensionales identificadas, lo cual servirá para fines de comparación en el análisis cualitativo de las secuencias generadas por los modelos.

3.3.2. Perfil de dominios de entrenamiento

Tal como se describió en la subsección 2.1.1, un modelo solo aprende información nueva desde su conjunto de entrenamiento, no desde el de validación. Por ello, se propone crear un **perfil de dominios de entrenamiento**, el cual consistirá en **una caracterización de todos**

los dominios conservados presentes en las 950 secuencias de proteínas del conjunto de entrenamiento. Posteriormente, este perfil podrá ser utilizado en el algoritmo de evaluación para determinar el porcentaje de las secuencias generadas que contienen los dominios correctos.

Para construir el perfil se hará uso de la herramienta web **Batch CD-Search** (ver subsección 2.3.4) a través de su API[27]. A pesar de que se puede interactuar con la herramienta mediante una interfaz gráfica[28], se prefirió el uso de la API debido a que permite un mayor control sobre la petición y la respuesta, a la vez que automatiza la generación del perfil de dominios.

En particular, la API permite buscar dominios conservados en lotes de secuencias en formato FASTA. Para ello se desarrolló un *script* en Python, el cual almacena en memoria el contenido de un archivo FASTA con las secuencias de entrenamiento y lo envía a través de una petición POST HTTP con los parámetros mostrados en el Listado 5.

Listado 5: Parámetros de la petición POST a la API de Batch CD-Search y ejemplo de ejecución.

```
import requests

url = "https://www.ncbi.nlm.nih.gov/Structure/bwrpsb/bwrpsb.cgi"
data = {
    "queries": fasta_sequences,
    "maxhit": 500,
    "smode": "auto",
    "tdata": "hits",
    "dmode": "rep",
    "qdefl": "false"
}
response = requests.post(url, data=data)
```

El detalle de los parámetros es el siguiente:

- `queries`: *string* con las secuencias a analizar en formato FASTA.
- `maxhit`: Número máximo de dominios conservados a retornar por secuencia. 500 es el valor por defecto.
- `smode`: Modo de búsqueda. `"auto"` significa que el programa elegirá si retornar un resultado precalculado o ejecutar RPS-BLAST directamente; pero, como se están analizando secuencias FASTA, se ejecutará el algoritmo de búsqueda sí o sí.
- `tdata`: Tipo de datos deseado en la salida. `"hits"` significa *domain hits* (*i. e.*, aciertos o coincidencias de dominios conservados).

[27] Documentación disponible en https://www.ncbi.nlm.nih.gov/Structure/cdd/cdd_help.shtml#BatchRPSBWebAPI.

[28] Acceso a la interfaz en https://www.ncbi.nlm.nih.gov/Structure/bwrpsb/bwrpsb.cgi.

- `dmode`: Modo para mostrar los datos en la salida. `"rep"` significa que, para cada región de cada secuencia de búsqueda, se mostrará el *hit* con mayor puntaje.
- `qdefl`: Booleano que indica si se debe incluir o no lineas de definición para las proteínas de búsqueda.

La respuesta de la API se almacena en la variable `response`. Si la petición es aceptada con éxito, la variable contendrá información en texto plano que incluye el identificador asignado al *job* correspondiente, tal como se ve en el Listado 6, en la segunda línea.

Listado 6: Ejemplo de contenido de la respuesta de la API de Batch CD-Search al aceptar una petición.

```
#Batch CD-search tool   NIH/NLM/NCBI
#cdsid  QM3-qcdsearch-357F51F4946FC1A7-323FE59A023C3AB8
#datatype       hitsConcise Results
#status 3       msg     Job is still running
```

El *script* desarrollado utiliza una expresión regular para extraer el identificador del *job* de la respuesta de la API y lo utiliza para ejecutar peticiones GET hasta que la API responda con el estado de éxito, recuperando los resultados de la búsqueda de dominios conservados. En el Listado 7 se muestra la expresión regular y un ejemplo de cómo se utiliza para extraer el `job_id` y, luego, hacer una petición GET.

Listado 7: Ejemplo de extracción de job_id y ejecución de petición GET a la API de Batch CD-Search.

```
import re

job_id_regex = re.compile(r"^#cdsid\s+([a-zA-Z0-9-]+)", re.MULTILINE)
job_id = job_id_regex.search(response.text).group(1)

url = f"https://www.ncbi.nlm.nih.gov/Structure/bwrpsb/bwrpsb.cgi?cdsid={job_id}"
new_response = requests.get(url)
```

Una vez detectado el estado de éxito, la API retorna el resultado de la búsqueda de dominios conservados, entregando un texto plano como el mostrado en el Listado 8, desde el cual se puede extraer la información contenida desde la línea 8 como una tabla de valores separados por tabulaciones.

Listado 8: Ejemplo de respuesta de la API de Batch CD-Search cuando se completa una búsqueda.

```
#Batch CD-search tool   NIH/NLM/NCBI
#cdsid  QM3-qcdsearch-3AD165D38C4B1785-9DCEDB594436700
#datatype       hitsConcise Results
#status 0
#Start time     2025-06-03T04:10:19     Run time        0:00:00:33
#status success

Query   Hit type        PSSM-ID From    To      E-Value Bitscore        Accession       ...
    Short name      Incomplete      Superfamily
Q#1 - >EDW83224.1_GK22747_[Drosophila_willistoni]       specific        269834  692     ...
    739     5.92866e-06     44.0733 cd14686       bZIP      -      cl21462
Q#2 - >EGY20823.1_hypothetical_protein_VDAG_02347_[Verticillium_dahliae_VdLs.17]        ...
    superfamily     473870  360     414   7.03361e-06      43.286  cl21462 bZIP ...
    superfamily         -       -
Q#2 - >EGY20823.1_hypothetical_protein_VDAG_02347_[Verticillium_dahliae_VdLs.17]        ...
    specific        395120  5       26    6.74275e-05      39.9738 pfam00172Zn_clus N ...
          cl00068
Q#3 - >CCD82052.1_hypothetical_protein_Smp_080420_[Schistosoma_mansoni] specific        ...
    269844  368     426     7.90821e-15  68.7609  cd14696 bZIP_Jun         - cl21462
Q#3 - >CCD82052.1_hypothetical_protein_Smp_080420_[Schistosoma_mansoni] superfamily     ...
    461108  39      201     4.31784e-06  47.5992  cl24159 Jun superfamily N -
```

Cada fila de la tabla retornada por la API contiene un *domain hit* con la siguiente información:

- `Query`: En el formato `Q#N - XXXXXX` donde `N` indica el número ordinal de secuencia de búsqueda y `XXXXXX` es el encabezado FASTA de la secuencia.
- `Hit type`: Tipo de *hit*, el cual representa distintos niveles de confianza y alcance.
- `PSSM-ID`: Identificador único de una matriz de puntuación específica para el dominio encontrado.
- `From`: Aminoácido de inicio del alineamiento.
- `To`: Aminoácido final del alineamiento.
- `E-Value` y `Bitscore`: Ver ecuación 6 en la página 22.
- `Accession`: Identificador de acceso del *hit*, el cual puede ser un modelo de dominio o una superfamilia.
- `Short name`: Nombre corto de un dominio conservado
- `Incomplete`: Indica qué extremos del *hit* están incompletos, en caso de que se trate de un *hit* parcial.
- `Superfamily`: Indica la superfamilia a la que pertenece el *hit*, si aplica.

Recuperados los resultados, el *script* procesa los datos, agrupando los *domain hits* repetidos con sus respectivos códigos de acceso (`Accession`) y nombres cortos (`Short name`) y contando la frecuencia relativa de cada uno. Luego, se calculan también los promedios y

desviaciones estándar de los estadísticos *bit score* y *e-value* para cada dominio conservado identificado, ya que son los estadísticos de significancia de los *hits*. Finalmente, toda esta información es almacenada en un archivo JSON para su posterior utilización, el cual contiene documentos que usan los códigos de acceso como llaves. En el Listado 9 se observan dos ejemplos de dominios conservados encontrados en el formato final; notar que dominios con desviaciones estándar nulas indican que dicho dominio solo se encontró una vez entre todas las secuencias de búsqueda

Listado 9: Dos dominios del perfil de dominios de los transportadores de glucosa en el formato JSON correspondiente.

```
{
    "COG0654": {
        "Short name": "UbiH",
        "bitscore_promedio": 132.369,
        "bitscore_std": null,
        "evalue_promedio": 1.29889e-33,
        "evalue_std": null,
        "frecuencia_relativa": 0.0010235415
    },
    "COG2814": {
        "Short name": "AraJ",
        "bitscore_promedio": 102.9375,
        "bitscore_std": 2.9960114319,
        "evalue_promedio": 3.50957375e-23,
        "evalue_std": 4.88958152e-23,
        "frecuencia_relativa": 0.0020470829
    },
    ...
}
```

3.3.3. Optimización de hiperparámetros para el ajuste fino

Existen varias técnicas para encontrar los hiperparámetros que optimizan el desempeño de un modelo. Sin embargo, realizar esta búsqueda en el caso de los LLMs es muy costoso debido a la gran carga computacional que supone su ejecución; hacer muchas pruebas distintas se vuelve intratable si no se cuenta con los recursos necesarios. Por ello, **se definió que cada modelo será ajustado con cuatro combinaciones distintas de hiperparámetros, de entre las cuales se elegirá la que obtenga un mejor puntaje para realizar la evaluación final de los modelos**.

Dado que los tamaños de los conjuntos de entrenamiento y de validación quedaron fijados, se decidió variar los valores de tres hiperparámetros relevantes para los tres PLMs elegidos: ***batch size* efectivo**, **tasa de aprendizaje** y **número de *epochs***. Esta decisión se fundamenta en

que se trata hiperparámetros que, si son bien optimizados, pueden permitirle a los modelos obtener un aprendizaje relevante a partir de conjuntos de datos reducidos.

Así, tomando en consideración la arquitectura de cada PLM, se definieron las cuatro combinaciones de hiperparámetros a probar manualmente con cada modelo, tomando los siguientes nombres: «**conservadora**», «**moderada**», «**agresiva**» y «**balanceada**». Los hiperparámetros conservadores utilizan un *batch size* efectivo y una tasa de aprendizaje pequeños, pero un gran número de *epochs*, promoviendo extraer la mayor cantidad de información posible a la vez que se evita el sobreajuste. Por otro lado, los hiperparámetros agresivos utilizan un *batch size* efectivo y una tasa de aprendizaje grandes, combinado con menos *epochs*, lo cual está bien para acelerar el entrenamiento, pero con el riesgo de una convergencia inestable. Los hiperparámetros moderados están a medio camino entre los primeros dos, mientras que los hiperparámetros balanceados combinan características de los tres anteriores.

Para los tres modelos se probaron tasas de aprendizaje en el mismo orden de magnitud, $\mathbf{10^{-5}}$. Este valor se justifica por el tamaño reducido del conjunto de entrenamiento y el hecho de que se trata de procesos de ajuste *fino*, *i. e.*, se busca que las actualizaciones de los parámetros sean suaves para que no exista un olvido catastrófico del conocimiento general del lenguaje de las proteínas obtenido en el preentrenamiento. Además, sus valores escalan de forma aproximadamente proporcional al *batch size* efectivo.

En el caso del *batch size*, este depende directamente de la memoria VRAM disponible. Si bien es posible simular uno de mayor tamaño con la utilización de pasos de acumulación de gradiente, excederse con ellos va en directo detrimento de la convergencia del modelo. Por ello, se probó el máximo *batch size* posible con cada modelo y se agregaron pasos de acumulación de gradiente según fuera necesario para alcanzar el *batch size* efectivo propuesto. De esta forma, ProGen2 y ProtGPT2, con una cantidad similar de parámetros entrenables, aceptaron un *batch size* máximo de 4 secuencias; por otro lado, ProLLaMA, con una cantidad menor de parámetros entrenables, aceptó sin problemas un *batch size* de 32, por lo que no utilizó pasos de acumulación de gradiente.

Por otro lado, se definió una gran cantidad de *epochs* en los tres modelos debido a la necesidad de extraer la mayor cantidad de información posible de las 950 secuencias de entrenamiento. Si se hubiese contado con un conjunto de entrenamiento de mayor tamaño, la cantidad de *epochs* requeridas hubiese disminuido.

Adicionalmente, el modelo ProGen2, en su *script* de ajuste fino, definía un número fijo de 400 *warmup steps*. Este valor es muy alto para el ajuste fino, ya que se ejecutarán menos pasos en el entrenamiento. Por ello, se decidió mantenerlo fijo en 30 *warmup steps*, para no ralentizar la convergencia y fijar la atención en los otros hiperparámetros. ProLLaMA, por otro lado, utiliza una proporción por defecto de *warmup steps* de 0.03 del total de pasos, por lo que se dejó fijo dicho valor. ProtGPT2 no utiliza *warmup steps*.

A continuación, se presentan tres tablas con las configuraciones de hiperparámetros propuestas para el ajuste fino de cada PLM seleccionado. Notar que ProLLaMA (ver Tabla 5)

utiliza una tasa de aprendizaje mayor en su combinación **agresiva** debido a que cuenta con menos parámetros entrenables, lo cual también explica que allí se utilicen solo 20 *epochs*. Por otro lado, tanto ProLLaMA como ProGen2 (ver Tabla 6) pueden ver favorecido su desempeño con un mayor número de *epochs* en su configuración **conservadora** debido a sus arquitecturas, las cuales involucran técnicas para condicionar la generación autorregresiva, a diferencia de ProtGPT2 (ver Tabla 4).

Tabla 4: Configuraciones de hiperparámetros para el ajuste fino de ProtGPT2.

Configuración	***Batch size* efectivo**	**Tasa de aprendizaje**	**N° *epochs***
A: Conservadora	8	1×10^{-5}	40
B: Moderada	16	3×10^{-5}	30
C: Agresiva	32	5×10^{-5}	20
D: Balanceada	16	2×10^{-5}	40

Tabla 5: Configuraciones de hiperparámetros para el ajuste fino de ProLLaMA.

Configuración	***Batch size* efectivo**	**Tasa de aprendizaje**	**N° *epochs***
A: Conservadora	8	1×10^{-5}	50
B: Moderada	16	3×10^{-5}	40
C: Agresiva	32	7×10^{-5}	20
D: Balanceada	16	2×10^{-5}	50

Tabla 6: Configuraciones de hiperparámetros para el ajuste fino de ProGen2-medium.

Configuración	***Batch size* efectivo**	**Tasa de aprendizaje**	**N° *epochs***
A: Conservadora	8	1×10^{-5}	50
B: Moderada	16	3×10^{-5}	40
C: Agresiva	32	5×10^{-5}	25
D: Balanceada	16	2×10^{-5}	50

3.3.4. Algoritmo de evaluación

Una vez ajustado un modelo, se procede a generar una cantidad determinada de secuencias con él utilizando los hiperparámetros de inferencia óptimos por defecto de cada modelo. En particular, se propone **generar 150 secuencias con cada uno y filtrar las 50 mejores según su perplejidad**. La justificación de esta cantidad de secuencias a generar tiene relación con el tiempo que toma la ejecución de inferencia en los modelos más grandes. Además, la autora

de ProtGPT2 mencionó como «regla de oro» la filtración del tercio de menor perplejidad de entre un conjunto de secuencias de proteínas generadas.[29]

El algoritmo de evaluación propuesto fue desarrollado en un *script* Python y recibe como entradas los siguientes parámetros:

- `train_sequences_path`: Ruta al archivo FASTA con las secuencias de entrenamiento.
- `domain_profile_path`: Ruta al archivo JSON con el perfil de dominios de entrenamiento.
- `generated_sequences_path`: Ruta a un archivo FASTA con las secuencias generadas por un modelo específico, filtradas por perplejidad.

Como parte del algoritmo, se ejecutará **BLAST** (ver subsección 2.3.3) utilizando como consulta las 50 secuencias filtradas y como objetivo las 950 secuencias de entrenamiento. De aquí, se obtendrán distintos alineamientos de secuencias (*i. e.*, *hits*), cada uno con distintas métricas, entre las que se destaca el porcentaje de similitud. Se utilizará el **porcentaje de similitud promedio** de todos los *hits* como una medida de sobreajuste; *e. g.*, si la medida fuera mayor a 90 %, significaría que, en promedio, cada *hit* entre una secuencia generada y una de entrenamiento presenta un 90 % de similitud, evidenciando que el modelo está memorizando más que generalizando. La ejecución de BLAST se realiza a través de su herramienta de línea de comandos `blastp`, la cual permite realizar alineamientos de secuencias de proteínas de forma local con la instalación de los binarios correspondientes.[30] En particular, la llamada a `blastp` se ejecuta con los siguientes parámetros:

- `-query generated_sequences_path`
- `-subject train_sequences_path`
- `-evalue 0.05`: Umbral de significancia por sobre el cual no se reporta un *hit*.
- `-word_size 3`: Cantidad de aminoácidos iniciales para encontrar alineamientos.
- `-matrix BLOSUM62`: Matriz de sustitución de aminoácidos estándar.
- `-gapopen 11`: Penalización por abrir un *gap*.
- `-gapextend 1`: Penalización por extender un *gap*.
- `-outfmt 10 delim=@`: Formato de la salida del programa, en este caso una tabla de valores delimitados por el carácter `@`.

[29] Ver discusión en https://huggingface.co/nferruz/ProtGPT2/discussions/38#65b7c3cc9d26b3b15da93792.

[30] Enlaces de descarga en https://blast.ncbi.nlm.nih.gov/doc/blast-help/downloadblastdata.html.

Posteriormente, se ejecutará **Batch CD-Search para encontrar dominios conservados en las 50 secuencias generadas** utilizando la API correspondiente, siguiendo el mismo procedimiento que para la búsqueda de dominios en las secuencias de entrenamiento (ver subsección 3.3.2), con la diferencia de que los resultados se almacenarán en un *data frame* para luego ejecutar la comparación con los dominios del perfil de dominios.

Para comparar los *domain hits* de las secuencias generadas con aquellos presentes en el perfil de dominios de entrenamiento, el algoritmo calculará un puntaje por secuencia que cumplirá con la siguiente condición:

$$\textbf{Puntaje}(\text{secuencia}_i) = \begin{cases} 0 & \text{si no hay } \textit{hits} \text{ de DCs coincidentes,} \\ \sum_{j=1}^{H} \textbf{HitScore}(\text{e-value}_j) & \text{si hay } H > 0 \textit{ hits} \text{ de DCs coincidentes.} \end{cases} \tag{7}$$

Donde la función **HitScore** aplica logaritmo negativo al *e-value* del *hit* j y lo transforma a una escala lineal, donde un *e-value* menor a 10^{-50} se considera un *hit* seguro y uno mayor a 10^{-2} se descarta. El límite inferior se estableció en dicho valor debido a que algunos *e-values* pueden volverse muy pequeños, algunos con órdenes de magnitud de 10^{-100}. Estadísticamente hablando, desde cierto umbral se puede considerar que *e-values* tan pequeños son equivalentes debido a su cercanía al cero. De esta forma, definiendo $L_{max} = -\log(10^{-50}) = 50$ y $L_{min} = -\log(10^{-2}) = 2$ y siendo $L_j = -\log(\text{e-value}_j)$, se tiene:

$$\textbf{HitScore}(L_j) = \begin{cases} 1 & \text{si } L_j > L_{max} \\ \frac{L_j - L_{min}}{L_{max} - L_{min}} & \text{si } L_{min} < L_j < L_{max} \\ 0 & \text{si } L_j < L_{min} \end{cases} \tag{8}$$

que es la función que se utilizará para el cálculo del puntaje de cada secuencia generada. Finalmente, **el puntaje global de un modelo estará dado por la media aritmética de los puntajes obtenidos por cada secuencia generada por este**. Por lo tanto, para N secuencias generadas por un PLM ajustado, se tiene:

$$\textbf{Puntaje global} = \frac{1}{N} \sum_{i=1}^{N} \textbf{Puntaje}(\text{secuencia}_i) \tag{9}$$

El resumen del método de evaluación desarrollado se presenta como pseudocódigo en el Algoritmo 1, donde el puntaje global es retornado en la variable `model_score`.

Algoritmo 1: Algoritmo de evaluación de un PLM.

Datos: `train_sequences`, `domains_profile`, `generated_sequences`

`n_similarity_hits`, `mean_similarity` ← ExecuteBlastp(*generated_sequences*, *train_sequences*);
`job_id`, `n_sequences` ← BatchCDSearch(*generated_sequences*);
`retries` ← 0;
mientras CheckStatus(*job_id*) $\neq$ *listo* **hacer**
    Esperar mín$(10 \times 2^{\texttt{retries}}, 300)$ segundos;
    `retries` ← `retries` $+1$;
fin
`domain_hits` ← RetrieveResults(*job_id*);
`total_domain_hits` ← Número de registros en `domain_hits`;
`results_dict` ← Un diccionario para agregar resultados por secuencia;
para *cada hit en domain_hits* **hacer**
    `seq_key` ← Identificador FASTA de la secuencia generada que contiene el `hit`;
    `results_dict[seq_key]` ← `Accession` y `E-Value` del `hit`;
fin
`correct_domain_hits` ← 0;
`model_score` ← 0;
`sequences_scores` ← Una lista vacía;
para *cada resultado por seq_key en results_dict* **hacer**
    `sequence_score` ← 0;
    para *cada Accession y E-Value en resultado* **hacer**
        si *el hit con Accession está en el domains_profile* **entonces**
            `correct_domain_hits` ← `correct_domain_hits` $+1$;
            `sequence_score` ← `sequence_score` $+$ HitScore(*E-Value*);
        fin
    fin
    Agregar a `sequences_scores` una tupla (`seq_key`, `sequence_score`);
    `model_score` ← `model_score` + `sequence_score`;
fin
`model_score` ← `model_score` $\div$ `n_sequences`;
`best_sequences` ← Las 5 mejores según su puntaje en `sequences_scores`;
`worst_sequences` ← Las 5 peores según su puntaje en `sequences_scores`;
devolver `n_similarity_hits`, `mean_similarity`, `total_domain_hits`, `correct_domain_hits`, `model_score`, `best_sequences`, `worst_sequences`

CAPÍTULO 4
VALIDACIÓN DE LA SOLUCIÓN

En este capítulo se exponen los resultados de la evaluación de los modelos seleccionados. Según lo propuesto anteriormente, se ejecutó el ajuste fino de cada modelo con cuatro combinaciones de hiperparámetros, *i. e.*, **se entrenaron doce modelos en total**. Por cada uno, se generaron alrededor de 150 secuencias que luego fueron filtradas y evaluadas mediante una **función de evaluación** para determinar su calidad. Esto permite concretar el objetivo específico **O3**, al evaluar los modelos según la calidad de las secuencias que generan. Finalmente, conforme al objetivo específico **O4**, se determina la plataforma más efectiva de entre los tres PLMs seleccionados para la generación de secuencias de proteínas *de novo*.

4.1. Entrenamiento de los modelos

El ajuste fino de los doce modelos se hizo sobre las 950 secuencias de entrenamiento previamente definidas, utilizando las combinaciones de hiperparámetros de la sección 3.3.3. A continuación se presentan detalles de las métricas obtenidas en su ejecución.

4.1.1. Duración del entrenamiento por modelo

El tiempo en minutos que tomó el ajuste fino de cada modelo se muestra en el gráfico de la Figura 11. Se observa una dependencia lineal entre el tiempo de entrenamiento y la cantidad de *epochs* utilizada en cada configuración de hiperparámetros, donde el tiempo absoluto está dado por otros factores relacionados con la arquitectura de cada modelo particular.

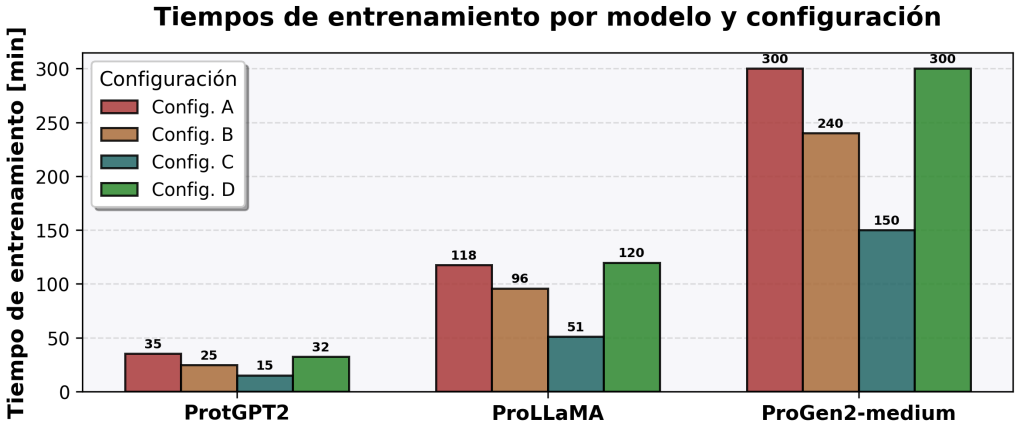


Figura 11: Tiempo de entrenamiento por modelo y configuración de hiperparámetros. Fuente: Elaboración propia.

4.1.2. Comportamiento de las curvas de pérdida

Dado que los tres modelos seleccionados siguen el objetivo de modelamiento causal del lenguaje de las proteínas, estos utilizan como función de pérdida la entropía cruzada (ver subsección 2.2.1). Se registraron los valores de las **pérdidas de entrenamiento (*train loss*) y de validación (*test loss*)** al final de cada *epoch* en los ajustes finos de ProLLaMA y de ProGen2-medium. En el caso de ProtGPT2 no fue posible efectuar este registro, a excepción del valor de las pérdidas al finalizar el entrenamiento.

Las curvas de pérdida por configuración de **ProLLaMA** se observan en la Figura 12 (ver definición de configuraciones en la Tabla 5). Se evidencia una clara divergencia entre ambas pérdidas en las cuatro configuraciones, donde incluso la pérdida de entrenamiento en las configuraciones B y D se acerca a cero. En teoría, esto se considera un indicativo de que el modelo está siendo **sobreajustado** al conjunto de entrenamiento, *i. e.*, está *memorizando* sus secuencias de entrenamiento y no logra predecir correctamente el siguiente *token* en secuencias que no ha visto, las de validación. En las cuatro configuraciones se observa un punto óptimo para la pérdida de validación alrededor del quinto *epoch*.

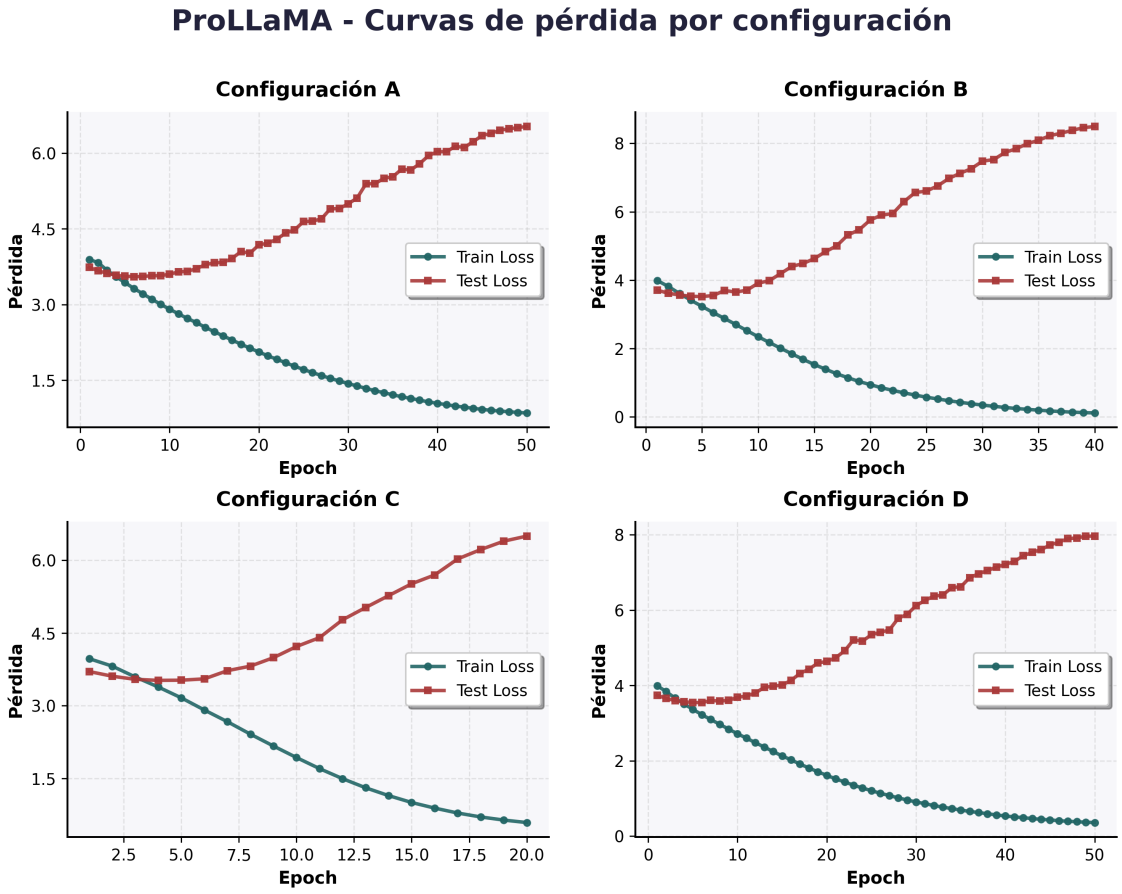


Figura 12: Curvas de pérdida por configuración de hiperparámetros de ProLLaMA. Fuente: Elaboración propia.

En el caso de **ProGen2-medium**, sus curvas de pérdida por configuración se encuentran en la Figura 13 (ver definición de configuraciones en la Tabla 6). Se observa una divergencia en las cuatro configuraciones, al igual que con ProLLaMA, pero con claras diferencias. En primer lugar, la pérdida de entrenamiento se satura mucho más rápido, disminuyendo de inmediato luego del primer *epoch* (*i. e.*, después de revisar todas las secuencias de entrenamiento por primera vez) y volviéndose asíntota al cero después del vigésimo *epoch* en todas las configuraciones. Por otro lado, la pérdida de validación es mucho más ruidosa e inestable, aunque mostrando una clara tendencia al alza en todas las configuraciones, excepto en C, donde la tendencia es más sutil. De igual forma, se observa un óptimo para la pérdida de validación entre los *epochs* 5 y 10.

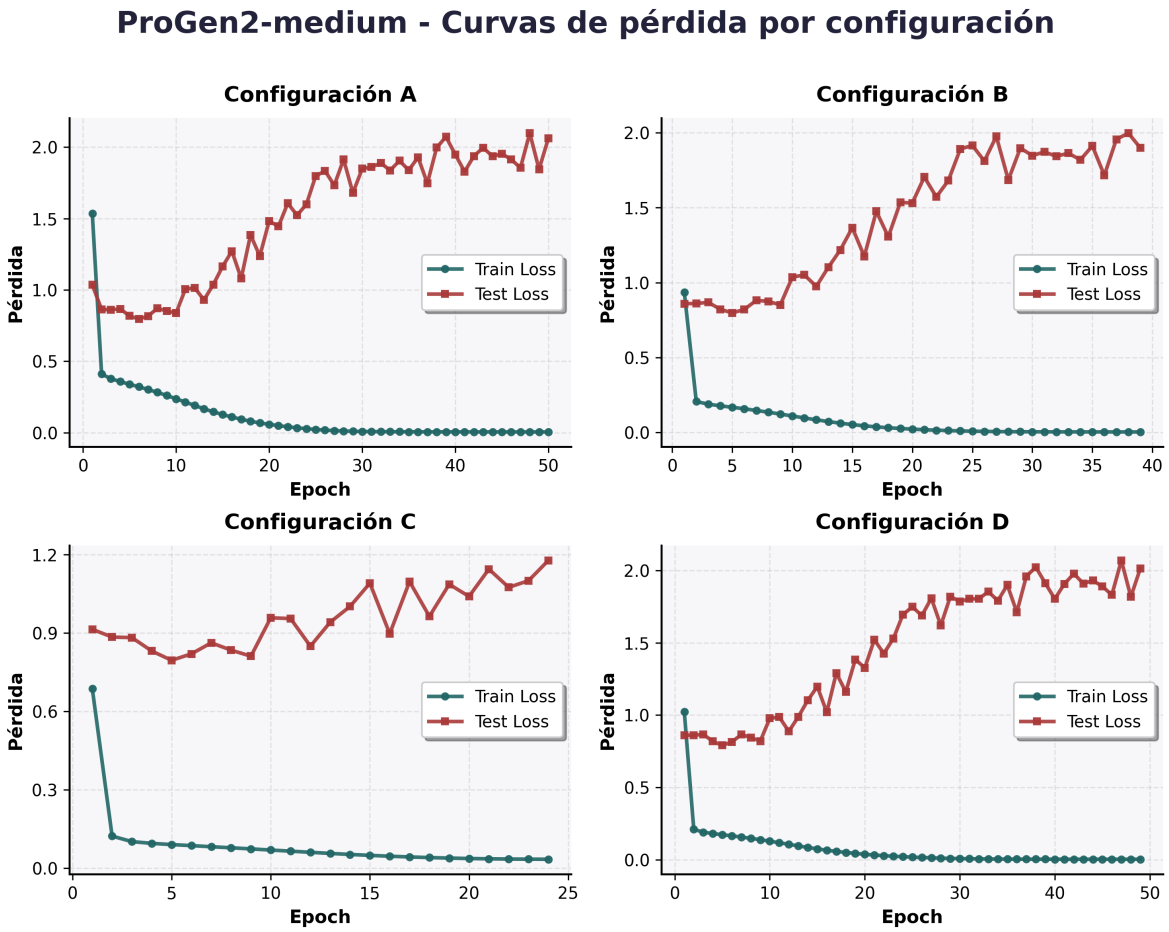


Figura 13: Curvas de pérdida por configuración de hiperparámetros de ProGen2-medium. Fuente: Elaboración propia.

Tal como se mencionó anteriormente, no fue posible registrar la evolución de las pérdidas en el caso de **ProtGPT2**. De todas formas, en la Tabla 7 se muestran los valores de las pérdidas al finalizar los entrenamientos con cada configuración de hiperparámetros del modelo (ver definición de configuraciones en la Tabla 4). De los datos presentados se puede notar que no hubo una disminución en la pérdida de entrenamiento al disminuir el número de *epochs*

desde la configuración A a la C, cosa que sí ocurrió en ProGen2 y ProLLaMA, mientras que la pérdida de validación es casi la misma en las cuatro configuraciones. Suponiendo que las curvas hayan tenido un comportamiento similar a las de los otros modelos, es posible que desde las 20 *epochs* ya se encontraran saturadas en todas las configuraciones, aunque queda pendiente revisarlo.

Tabla 7: Valores de pérdida finales de los ajustes con cada configuración de ProtGPT2.

Configuración	***Train loss***	***Test loss***
A	5.0575	5.2630
B	4.9657	5.2637
C	5.1710	5.2709
D	4.9880	5.2634

4.2. Evaluación de los modelos ajustados

Una vez efectuado el ajuste fino de los doce modelos, la siguiente fase consistió generar secuencias con cada uno de ellos, filtrar las mejores y aplicar el algoritmo de evaluación definido. Esto para determinar, por cada modelo, la capacidad de recuperar dominios conservados esperados y la capacidad de generar secuencias que no presenten una alta similitud con las secuencias de entrenamiento (ver Algoritmo 1).

4.2.1. Caracterización de las secuencias generadas

En primer lugar, se hizo **una caracterización estadística de las longitudes de las secuencias generadas** por cada modelo. Como referencia, en la Figura 14 se muestra un *boxplot* con la distribución de los largos de las secuencias en el conjunto de 950 secuencias de entrenamiento, donde se observa que la distribución es aproximadamente normal, con una gran presencia de *outliers* y, como es de esperarse, ningún largo supera los 1024 aminoácidos.

En la Figura 15 se muestran las distribuciones de los largos de las secuencias generadas por cada modelo. Se observa que, en el caso de **ProtGPT2**, las cuatro configuraciones generaron conjuntos con largos distribuidos de forma bastante uniforme; solo una contiene un *outlier* y corresponde a la configuración A. Las secuencias generadas por **ProLLaMA** son las que mostraron la mayor variabilidad en sus largos, y todas las configuraciones dieron con secuencias de más de 1024 aminoácidos, a pesar de que no habían secuencias así en los datos de entrenamiento; la configuración A fue la que presentó el caso más extremo. Finalmente, **ProGen2-medium** fue el modelo más capaz en reproducir distribuciones de largos de secuencias similares a la de los datos de entrenamiento; además, ninguna configuración generó *outliers* por sobre los 1024 aminoácidos.

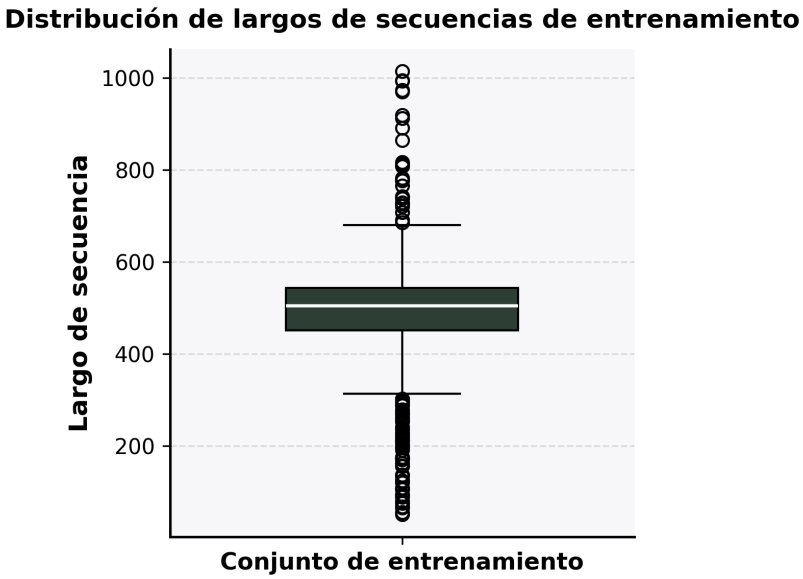


Figura 14: Distribución de largos de las secuencias de entrenamiento.
Fuente: Elaboración propia.

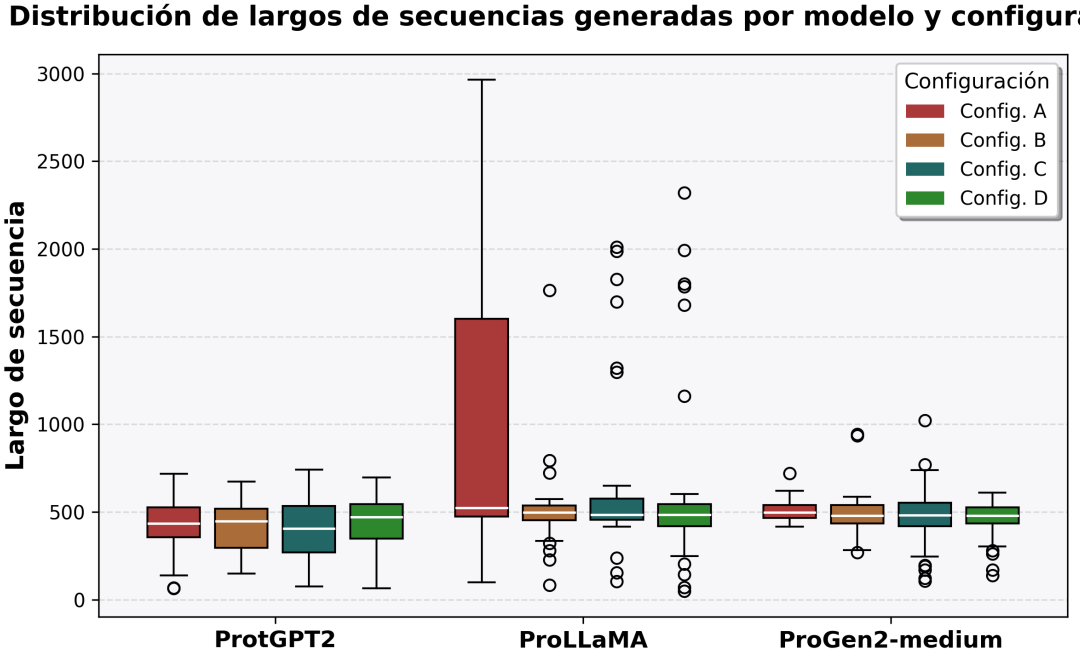


Figura 15: Distribución de largos de secuencias generadas por modelo y configuración.
Fuente: Elaboración propia.

4.2.2. Resultados de la aplicación del algoritmo de evaluación

A continuación, se presentan los resultados de la aplicación del algoritmo de evaluación sobre las secuencias generadas por cada modelo ajustado sobre cada configuración de hiperparámetros definida. De cada grupo de configuraciones, se selecciona como «ganadora» la que haya obtenido un mejor **puntaje global**, ya que este toma en consideración la presencia de dominios conservados pertenecientes al perfil de entrenamiento y su relevancia estadística (ver ecuación 9 y Algoritmo 1).

En el caso de **ProtGPT2**, se generaron 150 secuencias sin problemas por cada combinación de hiperparámetros, filtrándose las 50 mejores para la ejecución de la función de evaluación. Los resultados por configuración se resumen en la Tabla 8, donde «**Moderada**» fue la ganadora.

Tabla 8: Evaluación de cada configuración de hiperparámetros de ProtGPT2.

Resultado	**A: Conservadora**	**B: Moderada**	**C: Agresiva**	**D: Balanceada**
N° *domain hits* correctos	16	22	7	21
N° *domain hits* totales	64	56	61	57
Porcentaje dominios correctos	25.00 %	39.29 %	11.48 %	36.84 %
N° *hits* con secuencias de entrenamiento	6 424	11 023	2 675	9 959
Porcentaje similitud promedio	27.79 %	27.71 %	28.28 %	27.70 %
Puntaje global	0.246856	**0.428812**	0.099242	0.403051

En el caso de **ProLLaMA**, también se generaron 150 secuencias por cada ajuste, pero muchas de ellas contenían caracteres inválidos, ocurriendo que en sus cuatro configuraciones se obtuvieron menos de 50 secuencias válidas. El resumen de los resultados se presenta en la Tabla 9.

Tabla 9: Evaluación de cada configuración de hiperparámetros de ProLLaMA.

Resultado	A: Conservadora	B: Moderada	C: Agresiva	D: Balanceada
N° secuencias válidas	47 (31.3 %)	46 (30.7 %)	42 (28.0 %)	43 (28.7 %)
N° *domain hits* correctos	39	40	37	35
N° *domain hits* totales	54	50	42	45
Porcentaje dominios correctos	72.22 %	80.00 %	88.10 %	77.78 %
N° *hits* con secuencias de entrenamiento	13 061	17 262	15 876	13 289
Porcentaje similitud promedio	26.60 %	26.62 %	26.70 %	26.59 %
Puntaje global	0.429064	0.533552	**0.612430**	0.438726

Finalmente, en el caso de **ProGen2-medium**, la generación de 150 secuencias con cada combinación de hiperparámetros también dio como resultado algunas secuencias inválidas, aunque muchas menos que en el caso de ProLLaMA. Por esto, con cada combinación se generaron algunas secuencias de más para obtener el total de 150 válidas y así, luego, filtrar las 50 mejores según su perplejidad. El resumen de los resultados se muestra en la Tabla 10.

Tabla 10: Evaluación de cada configuración de hiperparámetros de ProGen2-medium.

Resultado	A: Conservadora	B: Moderada	C: Agresiva	D: Balanceada
N° *domain hits* correctos	51	51	29	51
N° *domain hits* totales	51	51	45	51
Porcentaje dominios correctos	100.00 %	100.00 %	66.44 %	100.00 %
N° *hits* con secuencias de entrenamiento	26 060	25 516	10 108	23 930
Porcentaje similitud promedio	27.49 %	27.78 %	28.23 %	27.93 %
Puntaje global	0.969627	0.968328	0.358885	**0.969937**

4.3. Análisis estructural de las secuencias generadas

En la sección anterior se pudo determinar el mejor modelo en términos de su capacidad de recuperar dominios conservados con una alta significancia estadística. En esta sección, para dar un cierre a la validación de la solución, se expondrán simulaciones de las estructuras tridimensionales de las mejores y peores secuencias generadas por cada modelo ajustado con su configuración de hiperparámetros óptima.

4.3.1. Selección de secuencias para simulación

Como se puede observar en la descripción del algoritmo de evaluación (ver Algoritmo 1), este retorna una lista con las cinco «mejores» secuencias y otra con las cinco «peores». El criterio utilizado para esta definición fue ordenar todas las secuencias generadas según su puntaje, el cual está dado por la presencia de dominios conservados correspondientes al perfil de dominios de entrenamiento y la significancia estadística de cada coincidencia. De esta selección, se extrajeron las dos mejores y las dos peores de cada modelo óptimo para simular sus estructuras tridimensionales. En la Tabla 11 se presenta una caracterización de las secuencias seleccionadas, mostrando sus puntajes asignados y los dominios conservados encontrados en ellas, cada uno identificado con su respectivo código de acceso y nombre común, mostrando en negrita aquellos con un *match* en el perfil de dominios. Como referencia, en el Anexo 5.1 se adjuntan las secuencias en formato FASTA.

Modelo	Calidad	Secuencia	Largo	DC identificado	Nombre común	Puntaje
ProtGPT2 (Config. B)	Mejores	>seq_0	476	**cl28910**	***Major Facilitator Superfamily***	1.000000
		>seq_3	423	**cl28910**	***Major Facilitator Superfamily***	1.000000
	Peores	>seq_43	429	COG2807	*Cyanate permease*	0.000000
		>seq_45	281	COG1028	*NAD(P)-dependent dehydrogenase*	0.000000
ProLLaMA (Config. C)	Mejores	>seq_2	1 827	**cl28910**	***Major Facilitator Superfamily***	1.000000
		>seq_6	450	**cl28910**	***Major Facilitator Superfamily***	1.000000
	Peores	>seq_0	2 011	pfam07690	*Major Facilitator Superfamily*	0.000000
		>seq_33	573	**cl28910** cl34453	***Major Facilitator Superfamily*** –	0.227710
ProGen2-medium (Config. D)	Mejores	>seq_8	562	**cl28910** **cl46767**	***Major Facilitator Superfamily*** ***Arc-like DNA binding domain***	1.010783
		>seq_0	610	**cl28910**	***Major Facilitator Superfamily***	1.000000
	Peores	>seq_20	452	**cd17316**	***Metazoan SV2***	0.893577
		>seq_29	260	**cd17316**	***Metazoan SV2***	0.841764

Tabla 11: Selección de mejores y peores secuencias generadas por cada modelo ajustado.

4.3.2. Metodología de simulación estructural y resultados de las simulaciones

Para ejecutar la simulación de las estructuras tridimensionales se utilizó **ColabFold**[31], un *notebook* abierto que se ejecuta en Google Colab para facilitar la realización de predicciones de estructuras proteicas utilizando **AlphaFold2** [Mirdita *et al.*, 2022, Jumper *et al.*, 2021]. Las secuencias se fueron simulando una a una utilizando todos los parámetros por defecto. Como salida, ColabFold entrega un archivo comprimido con distintos **archivos PDB**[32], correspondientes a distintas iteraciones de simulación.

Los archivos PDB entregados por ColabFold son ordenados según su el **puntaje pLDDT** (*predicted Local Distance Difference Test*) promedio asignado a la estructura simulada. Este puntaje mide la confianza local por aminoácido en la predicción de la estructura tridimensional, consistiendo en una escala que va de 0 a 100, de menor a mayor confianza. Se seleccionaron la iteraciones con mayor puntaje pLDDT promedio de cada secuencia y se graficaron con el uso del software abierto **ChimeraX**, el cual permite colorear las estructuras según sus puntajes pLDDT [Pettersen *et al.*, 2021].

Para tener una referencia estructural, en la Figura 16 se muestran cuatro ejemplos de proteínas naturales pertenecientes a la **familia de transportadores de glucosa**, cuyos códigos de acceso PDB son `6tha`, `6n3i`, `4pyp` y `4gby`.

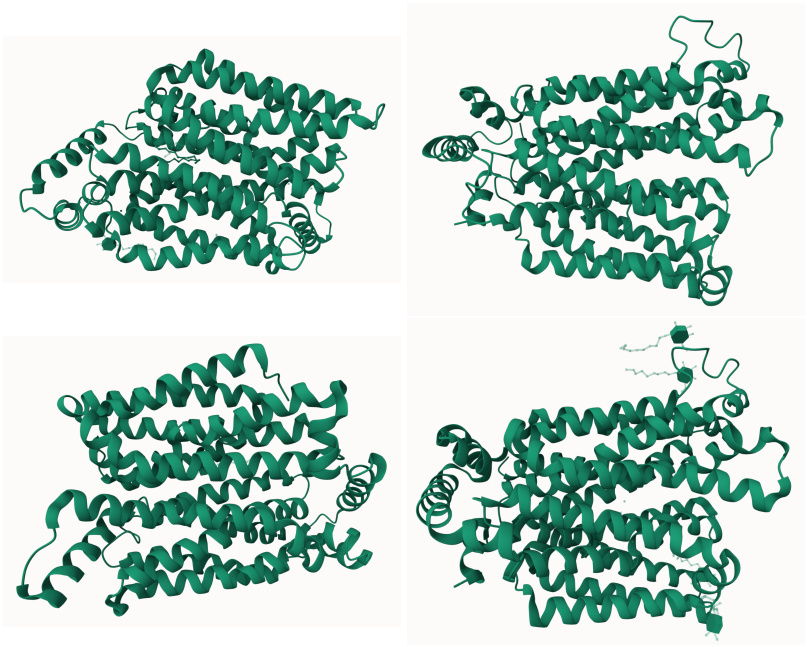
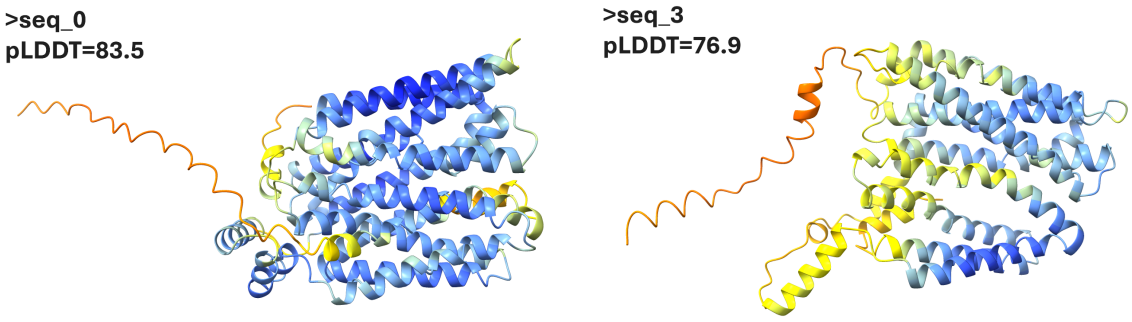


Figura 16: Estructuras tridimensionales de transportadores de glucosa naturales.
Fuente: Extraídas de la base de datos InterPro.

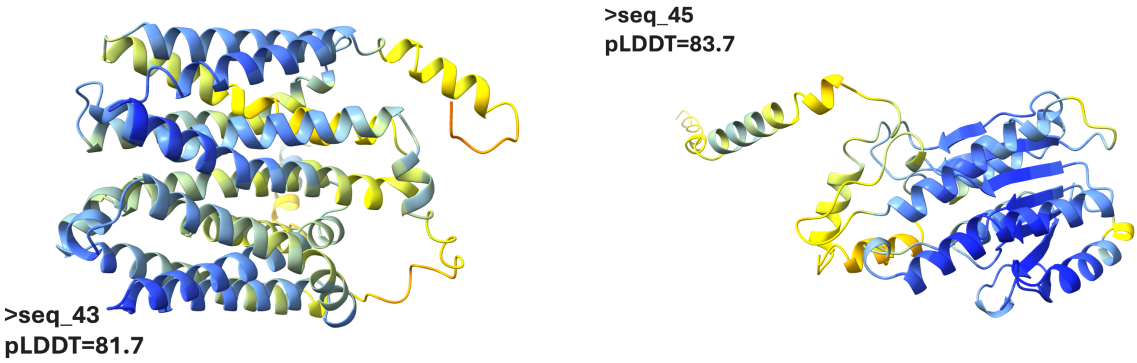
[31] Disponible en `https://colab.research.google.com/github/sokrypton/ColabFold/blob/main/AlphaFold2.ipynb`.

[32] Acrónimo de *Protein Data Bank*, formato de archivo de texto que describe las estructuras tridimensionales de moléculas para su representación gráfica en algún *software* de destino.

A continuación, se presentan los resultados de las simulaciones estructurales de las secuencias seleccionadas para cada modelo, las cuales corresponden a las presentadas en la Tabla 11. En la Figura 17 se muestran las secuencias generadas por ProtGPT2, en la Figura 18 se muestran aquellas generadas por ProLLaMA y en la Figura 19 se muestran aquellas generadas por ProGen2-medium. En todos los casos, la coloración azul indica un alto puntaje pLDDT y, mientras más corrido al rojo, peor pLDDT.

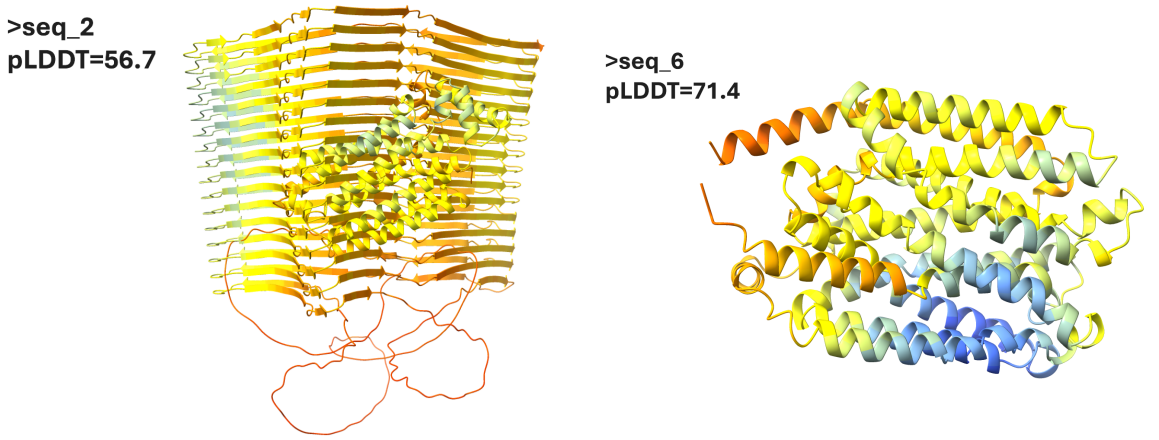


(a) Secuencias con mejor puntaje global de dominios.

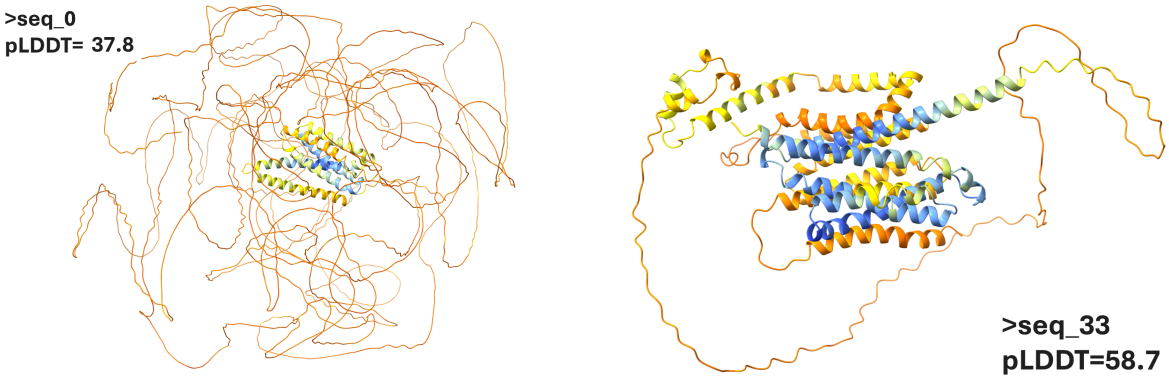


(b) Secuencias con peor puntaje global de dominios.

Figura 17: Simulaciones estructurales de secuencias generadas por ProtGPT2.
Fuente: Elaboración propia.

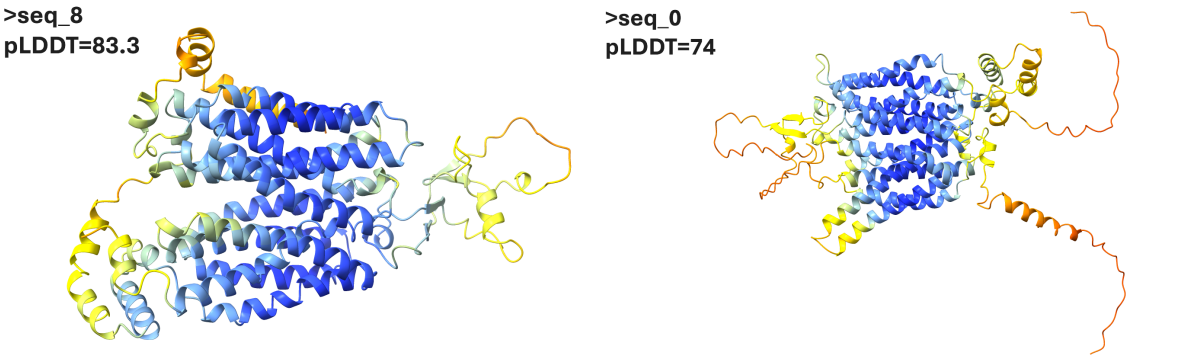


(a) Secuencias con mejor puntaje global de dominios.

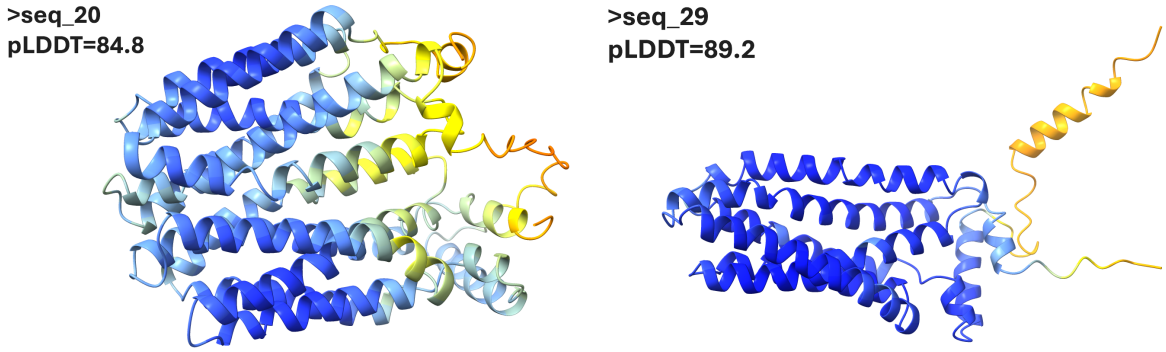


(b) Secuencias con peor puntaje global de dominios.

Figura 18: Simulaciones estructurales de secuencias generadas por ProLLaMA.
Fuente: Elaboración propia.



(a) Secuencias con mejor puntaje global de dominios.



(b) Secuencias con peor puntaje global de dominios.

Figura 19: Simulaciones estructurales de secuencias generadas por ProGen2-medium. Fuente: Elaboración propia.

Observaciones clave:

- No parece haber una correlación entre la confianza estructural y la confianza de los dominios conservados que fueron identificados en las secuencias simuladas.
- Las mejores secuencias de ProtGPT2 tomaron estructuras similares a las naturales. Sus peores secuencias tomaron estructuras ligeramente diferentes, correlacionándose con la presencia de dominios no pertenecientes al perfil esperado.
- La función de evaluación determinó que la secuencia 2 generada por ProLLaMA era la mejor en términos de confianza en su *hit* de dominio conservado. Sin embargo, la simulación estructural muestra que solo una parte de la secuencia se pliega en una estructura esperada, pero el resto conforma una estructura repetitiva que termina dominando la conformación completa.
- Las secuencias 2 y 0 de ProLLaMA tenían largos superiores a 1024 aminoácidos. En el caso de la secuencia 0, se observa una larga cadena desordenada que se «enreda» alrededor de la estructura central.

- Todas las secuencias generadas por ProGen2-medium contenían dominios conservados pertenecientes al perfil esperado. Las «peores» tuvieron menor puntaje solamente porque sus *hits* presentaron una menor relevancia estadística, pero todas presentan estructuras similares a las naturales.

4.4. Discusión y *benchmarking*

Esta sección tiene por objetivo discutir los hallazgos obtenidos a lo largo de la validación de los modelos de lenguaje de proteínas seleccionados, así como establecer una comparación que permita a los *stakeholders* decidir qué modelo utilizar para la ejecución de sus propias tareas de entrenamiento y generación de secuencias de proteínas *de novo*.

Como primera conclusión de la interpretación general de los resultados obtenidos, **quedó demostrada la importancia de optimizar los hiperparámetros de entrenamiento de un modelo de lenguaje antes de ejecutar ajuste fino sobre un conjunto de entrenamiento de tamaño reducido**. Basta comparar los resultados de las pruebas de concepto en la recuperación de dominios conservados esperados (ver Tabla 3) con los resultados de la evaluación efectuada para dar cuenta de ello.

Respecto a los resultados por modelo, es claro que **ProGen2-medium demostró el mejor desempeño**. Las razones detrás de ello son que fue el modelo mejor evaluado por el algoritmo propuesto, generando casi sin problemas 150 secuencias válidas, de entre las cuales las 50 mejores según su perplejidad, todas contenían al menos un dominio conservado correcto. Además, fue capaz de replicar una distribución de largos de secuencias similar a la de los datos de entrenamiento sin tener sobreajuste. Por otro lado, el excelente desempeño de este modelo se pudo observar desde el proceso de entrenamiento, donde la pérdida de entrenamiento disminuyó inmediatamente después de solo un *epoch*, para luego acercarse a cero rápidamente. Su única desventaja radica en el costo temporal que requiere su ejecución: fue el modelo que más tiempo tardó en entrenar e inferir secuencias, totalizando cinco horas para entrenar en su configuración de hiperparámetros óptima (config. D). Sin embargo, el tiempo de entrenamiento podría acotarse a la mitad estableciendo que el ajuste se detenga una vez que la pérdida deje de mostrar mejoras significativas, lo cual ocurrió aproximadamente desde el *epoch* 25 según el gráfico correspondiente en la Figura 13.

El modelo **ProtGPT2**, por su parte, **fue el de peores resultados** en su capacidad de generar un aprendizaje relevante respecto al conjunto de entrenamiento. No logró recuperar más de un 40 % de dominios correctos, por lo que obtuvo el peor puntaje de parte del algoritmo de evaluación. Lo único destacable de este modelo tiene relación con ser el más amigable en términos de requerimientos computacionales, pudiendo ejecutarse para inferencia incluso en una computadora personal con al menos 4 GB de VRAM. Sin embargo, si se tiene acceso a recursos computacionales más robustos, no tendría sentido utilizar este modelo.

Por otro lado, **ProLLaMA se encuentra en un punto intermedio**. Más del 88 % de los dominios identificados en sus secuencias generadas pertenecían al perfil de dominios de entrenamiento, aunque con una significancia estadística menor que ProGen2-medium. Además, tarda en entrenar menos de la mitad del tiempo que este último modelo. Su gran desventaja tiene relación con que las secuencias que genera son susceptibles a tener mucho ruido, generando motivos secuenciales indeseados y simulaciones estructurales poco confiables.

Un asunto que fue transversal a los tres modelos, en todas sus configuraciones de hiperparámetros, fue el **bajo porcentaje de similitud promedio**. Esto demuestra, desde un punto de vista biológico, que los modelos están generando secuencias *de novo*, al mismo tiempo que conservan los dominios que aprenden que son importantes para reproducir proteínas similares a las de la familia de entrenamiento.

4.5. Limitaciones del estudio y propuestas de mejora

Si bien este trabajo logró cumplir todos sus objetivos propuestos, es importante reconocer las limitaciones metodológicas y técnicas que condicionaron el alcance de los resultados. En esta sección se presentan dichas limitaciones y se presentan propuestas de trabajo futuro.

Una de las principales limitantes al inicio del desarrollo de la investigación tuvo directa relación con los tamaños de los conjuntos de datos que se requería utilizar para el entrenamiento de los modelos. Dada una falta de experiencia previa trabajando con grandes modelos de lenguaje, no fue inmediatamente obvia la necesidad de experimentar con conjuntos de mayor tamaño, ni la necesidad de optimizar los hiperparámetros acorde a las características de los datos de entrenamiento.

Luego, cuando se dilucidó la necesidad de experimentar con distintos hiperparámetros por modelo, apareció la limitante de los recursos computacionales disponibles. Esto incidió directamente en la selección de los PLMs a probar, incluyendo el hecho de que no se pudieron ejecutar las versiones más grandes (y probablemente de aún mejor desempeño) de ProGen2. Una vez ejecutados los entrenamientos, la generación de secuencias también se vio limitada por los recursos computacionales. En particular, para una evaluación más exhaustiva, se podrían haber generado aún más secuencias por modelo, pero eso hubiese requerido mantener los modelos ejecutando inferencia por más tiempo.

Por el lado del algoritmo de evaluación, este podría haber incluido otro tipo de métricas, pudiendo evaluar otras aptitudes químicas de las secuencias generadas que no fueron incluidas como parte del estudio, tales como predicción de solubilidad o toxicidad. Además, al ejecutar las simulaciones de las estructuras tridimensionales de las secuencias, se eligió utilizar AlphaFold debido a que se espera que, como son generadas por modelos entrenados sobre conjuntos de secuencias naturales, estas deberían tener homologías claras. Sin embargo, existen otros modelos que simulan estructuras tridimensionales basándose solo en la información secuencial, como **ESMFold**, que serían interesantes de probar en un futuro.

También cabe destacar que, si bien las secuencias de ProtGPT2 y ProGen2-medium fueron filtradas según su perplejidad, es sabido que esta métrica solo evalúa la capacidad del modelo para predecir correctamente el *token* siguiente dado una secuencia de entrada, lo cual no implica necesariamente una buena aptitud molecular. Adicionalmente, en el caso de ProLLaMA se aceptaron secuencias generadas con más de 1024 aminoácidos, lo cual puede haber sido un error ya que, luego, al aplicar la función de evaluación, se determinó como una de sus mejores secuencias a una de 1827 aminoácidos, la cual dio como resultado una estructura tridimensional anómala (ver `>seq_2` en la Figura 18).

Por otro lado, si se asume que estos PLMs han logrado aprender patrones presentes en el lenguaje de las proteínas, especialmente en su fase de preentrenamiento, cabe la posibilidad de que, al efectuar el ajuste fino para sesgar la generación de secuencias hacia proteínas con propiedades específicas, se generen secuencias que presenten dominios conservados con esas propiedades específicas, pero que no necesariamente hayan estado presentes en el perfil de dominios del conjunto con el cual se haya efectuado el ajuste fino particular. Esto es relevante, ya que el algoritmo de evaluación compara los dominios conservados identificados en cada secuencia generada con el perfil de entrenamiento, y si no hay un *match*, simplemente las va descartando, en vez de revisar si, quizás, hay una pertenencia a una familia proteica con las propiedades deseadas.

En la misma línea anterior, también podría ocurrir que una secuencia generada *de novo* no contenga ningún dominio conservado conocido pero que, aún así, presente características biológicas novedosas que merezcan ser evaluadas en mayor profundidad. Lamentablemente, en esto entra en juego la incerteza inherente a este ámbito, ya que investigar una proteína completamente nueva y desconocida, de la cual no se puedan obtener predicciones certeras, se convierte en una especie de apuesta, la cual puede ser costosa.

Finalmente, dado que ya se concluyó que el modelo con mejor desempeño fue **ProGen2-medium**, otra línea de investigación futura interesante sería experimentar con los parámetros de inferencia. Se explicó anteriormente que se utilizaron los parámetros por defecto para esta tarea, pero es sabido que, ajustándolos, se puede hacer que el modelo sea más o menos creativo en su generación, pudiendo dar con resultados distintos e interesantes.

CAPÍTULO 5
CONCLUSIONES

El presente trabajo tuvo como objetivo general analizar modelos de inteligencia artificial generativa que producen secuencias de proteínas *de novo* pertenecientes al estado del arte y evaluar su efectividad en base a métricas biológicamente relevantes. Se propusieron cuatro objetivos específicos, los cuales fueron abordados y completados a cabalidad, consolidando un proyecto que se espera sea un aporte significativo a la innovación biotecnológica en AQUIT, empresa dedicada a la investigación biotecnológica orientada al desarrollo de una acuicultura sustentable.

En la Introducción se expuso la motivación que da origen a este campo de investigación, destacando el potencial que tienen los grandes modelos de lenguaje para desentrañar el «lenguaje» de las proteínas. En el Capítulo 1, se definieron conceptos clave como el diseño racional de proteínas *de novo*, el enfoque generativo, las aplicaciones actuales de la IA generativa en este ámbito y las problemáticas emergentes, estableciendo además los objetivos de este trabajo. En el Capítulo 2 se delinearon los conceptos bajo los cuales se cimentó la solución propuesta, dividiendo su contenido entre la base informática y la base biológica. En el Capítulo 3 se comenzaron a abordar los objetivos, al presentar un estudio del estado del arte, la selección de modelos de lenguaje de proteínas y las pruebas de concepto relacionadas con su implementación. Asimismo, se definió la propuesta de solución con su diseño experimental respectivo y el desarrollo de un algoritmo de evaluación. Finalmente, en el Capítulo 4, se ejecutó la solución diseñada, la cual consistió en el ajuste fino de los tres PLMs seleccionados sobre un conjunto curado de secuencias, cada uno con cuatro combinaciones de hiperparámetros diferentes, y su evaluación respectiva. Los resultados fueron analizados cuantitativa y cualitativamente, siendo presentados en tablas y gráficos de autoría propia, permitiendo validar la solución propuesta.

Como en toda investigación emergente, el desarrollo de esta memoria de título no estuvo exenta de desafíos. Tal como se señaló al definir el problema, el uso de GenAI para la generación de proteínas *de novo* sigue siendo un ámbito reciente y cambiante. Constantemente proliferan nuevos modelos, todos prometiendo superioridad respecto a sus predecesores, pudiendo generar escepticismo entre los investigadores que buscan resultados confiables y reproducibles.

En este trabajo se puso el foco en los modelos de lenguaje de proteínas, dada su ventaja comparativa frente a otras arquitecturas generativas como los modelos de difusión, la cual está dada por el éxito comprobado de los *transfomers* y la abundante disponibilidad de datos secuenciales en bases de datos de proteínas abiertas. No obstante, el gran inconveniente de los modelos de lenguaje es que aquellos de mejor desempeño son altamente demandantes en términos computacionales, lo que impone una barrera de entrada significativa para muchos grupos de investigación: solo para ejecutar experimentación básica se requiere acceso

a recursos especializados y, eventualmente, una inversión económica considerable.

A esto se suma el gran cuello de botella que representa la síntesis en laboratorio de las secuencias *de novo* generadas. Como se discutió en el Capítulo 1, los procesos involucrados son sumamente costosos y son transversales al diseño y producción de proteínas *de novo*, *i. e.*, cualquier método generativo se enfrenta a ello. Una proteína *de novo* generada y simulada computacionalmente puede verse sumamente prometedora, pero no se puede asegurar con plena certeza que cumplirá con la estructura y función deseadas a menos que se sintetice y se experimente con ella en el mundo físico. Esta brecha estructural entre el mundo *in silico*[33] y el mundo experimental sigue siendo uno de los principales desafíos para el diseño de proteínas. En este sentido, el presente trabajo se alineó con una tendencia creciente en biotecnología: avanzar en marcos computacionales robustos que prioricen la eficiencia, la interpretabilidad y la reducción de costos en etapas previas a la experimentación física.

No obstante lo anterior, el futuro de los modelos de lenguaje de proteínas aplicados al diseño de secuencias *de novo* es claramente promisorio. Nuevos avances en las tecnologías y algoritmos subyacentes ocurren a un ritmo acelerado. Por ejemplo, algunos autores se han cuestionado si escalar el tamaño de los PLMs es realmente necesario para mejorar su desempeño, o si, quizás, hay otros aspectos con espacios de mejora que puedan permitir alivianar estos modelos [Fournier *et al.*, 2024], lo que podría llevar al desarrollo de arquitecturas más eficientes y accesibles.

Por otro lado, dada la naturaleza opaca de los *transformers*, existe una necesidad urgente de avanzar en herramientas de explicabilidad. Estos métodos pueden permitir identificar las relaciones entre fragmentos de secuencia, motivos estructurales y funciones finales. Esto no solo es importante porque podría promover el desarrollo de una eventual «ciencia teórica de las proteínas», si no que también permitiría a todos los científicos del área tener más confianza en los resultados de los modelos generativos.

En síntesis, en este trabajo no solo se entregó una evaluación comparativa y sistemática de la aplicación de grandes modelos de lenguaje a la generación de proteínas *de novo*, sino que también se propuso un marco metodológico replicable, que puede servir tanto a futuras investigaciones como a proyectos industriales de biotecnología. Para AQUIT, en particular, representa una base concreta para explorar soluciones biotecnológicas innovadoras orientadas al cumplimiento de sus propios objetivos mediante la aplicación concienzuda de la inteligencia artificial generativa.

[33] Locución latina que significa «en el silicio». En biología se refiere a cualquier simulación computacional que aún no ha sido validada en el mundo físico.

ANEXOS

5.1. Selección de secuencias para simular estructuras tridimensionales

A continuación se presentan en formato FASTA las secuencias utilizadas en el análisis estructural de la sección 4.3.1.

Listado 10: Mejores y peores secuencias generadas por ProtGPT2 - Configuración B.

```
>seq_0
MTRRRSSRRLSGPAASAPPPLPPSPSRHLPLQVLVTAVLGNVLQGWDNAVIAGAVLFIQR
EFGVGEEFTELLVGLSITSICVIIVSPISGSLADRLGRKTIILASGAIYIFSGLVMAFSP
SYWCLIIGRALGGLGVGYALMIAPVYTAEISPPHTRGALVSLPEVFINTGILLGYVSNFA
FAHVPSGWRWMLGLAAAPGVLVTVGILSMPETPRWLIMKHRNEEALSVLKRTSDVAEAED
RLRQIKLAVKSPSSQPWSALLAAPSVRRTVIMAAVLQAFQQFTGINTVMYYSPTIVQMAG
FNSNKDALLAVALLGVVNTLMTFPAIWTVERFGRRTLLLTSLGGVFLALGLLAAGFWFQL
GQNPGGGTALALIGIMLYVAGHASGLGTVPWQQSELFPLSVRALGCGVATTVNWTCNAAI
SLTFLSLEEAVTPWGTFALYAGLNVLGALFALICVPETRGRSLEEMDQVYKGARGA
>seq_3
MSSPWLAWCGRFSRPDSGLLLWGIILCTVGAFTQGLAANVHSMVFFRAFQGIGGSAFLTI
GGGIIVDLFAHEERGKAVGLYYMGPLIGPISGPLIGGFIAEHLGWKWIFWVIAILFGIVQ
ICGSFLMRETYAPIIIAKRQKRLQQESNESRYWSKIERDNLSVKSMYLTMLVRPLKMLML
DPVIAILSLYMSFIYGLLYFMFAAFPIVFQKVHGWSVGVAGLALIPIGVGSVLGAAITCY
TDNRMHKLAKKHEMPEFHLPLLPVVVPLAGLALIAFGWTAEQYVHWIMPIIGTAVYGAGT
VLCFLSVQTYLVDTFVKASSSAIAANSVFRSFFAAIFPLFGLQMYNNLGIHWASSLLGFI
SLALIPAPWILYRYGPRLRAKSEYAQPHNPSANPTTAAQASGEGHDNEKAPDQKARESEA
SQA
>seq_43
MAEADLSLKARAVRKLEPKARVIIWAGVVVFVADMFLYGSYATKAPFVPALEREFGLGYQ
DFALLAVMNALAAVLLALPAGRLIDRLGPVAAALVGGVLAAAGVSFAAMSSSYTGFLIGF
VLLGVGMSTANIALPPLVKRWFPDRLATINAVYTLAFALGALAAGALVPPTVERFGDWRG
GYAAMGVIAAVTGVVWAVLSARFTASTPVRAAQGGAAPAEPVGIGALVAHPAWGVLTGYF
FLMGLGLYSFSLWLPAFLTESGHTGTVAGLQTAVATGLGIIGHLVLPAISWRMENRRVFI
AVLLLLAVFGYAALTLTQSPVGAWLVAAAIGALTFVVPLTLLALPALVTTPRQAGVVSGV
TNAGAYLGMSAGPMIVGLLSDRTGGFVAPMWILVCAAAALLVVVLLSPPRTRARRAVSAS
SGGSEPALA
>seq_45
MKILIVGATGGIGREIATACLAKGHTVVGLARNAAKTSVPAEFIPVSVTDAGSLATALAQ
EHPGGIDAIVCAAGVNQFDAIADPAQATAIIDTNLAGTYNVLRAAIAAGMGRCVVTGSHA
WYRPVPDFPKTWGMPIYNVSKAALNAWSRNLSIDRPDGMLVNSVSPGYIETELNGAFFAA
SGLEGPEAFARHYAEMVEPHGLKRFAQADEVADAIAFAVSAPAQSTTGAVFDVTGAHTVL
NVRGELLRPLRRAQEAAAAKAQSAAPPTPAAQPSAEPAPAQ
```

Listado 11: Mejores y peores secuencias generadas por ProLLaMA - Configuración C.

```
>seq_2
MKRFFTTAALASMAGTLIFGYDTGVMSGLLTMPAFVRDFYPAMDHESFMEAARTVGLCSALLSAGAAIHSWIGDKLGRRRLFLWSMYLISGS
LLQTACVSRAQLVIAGRILAGFAVGGLTATIPVFIAEVAEYQRGKVTSIYQTFWFVTNLGILWAQWIACRYSADLSWRFPLAFQCIIPSITL
MVALMLLPETPRWLIAQNRLSEARYIVAKLRNIPEADDSVRKSLGEVSFCLDEESSSYLEIKQAFSTLFENRSFRRTMLIGTSLFDSFTAVL
VVMTFSPQMQAVVISLDLWNAATAINYCYFVGCAIAACFLAMKLGRRALWMFSFVCFCVTTICLGTGSVLSGETIHLPIPGNTTHNELKEAN
IDVEAPSKSDSCSRNGTEEVVDTDNDGCTEREDGCPPTSGNETTYNSCVPPPNCDYCGACTEGWTGNWSEACEYTDIGCVPMDDPNNPYTWD
NYCKNHTYYMANVSNFSSLVCDDRYSWVFPDVTCENTFEWDYNFCCKNPWYYDDTRFNEPCNRDLWYDQFYECTTPMWYYYPAVANFNNSCL
AANGCTDAKLYWNSTDMQSWDYDLCRNRTYYMANVSNFSSLVCDDRYSWTWPDVTCENTFEWDYNFCCNAPWYYYDAATFNNSCMGANDCAD
AKLYWNSTDMQSWDYDLCRNRTYYMANVSNFSSLVCDDRYSWTWPDFTCENTFEWDYNFCCNAPWYYYDAATFNNSCMGANDCATGKLYWNS
TDMQSWDYDLCRNRTYYMANVSNFSSLVCDDRYSWTWPDFTCENTFEWDYNFCCNAPWYYYDAATFNNSCMGANDCATGKLYWNSTDMQSWD
YDLCRNRTYYMANVSNFSSLVCDDRYSWTWPDFTCENTFEWDYNFCCNAPWYYYDAATFNNSCMGANDCATGKLYWNSTDMQSWDYDLCRNR
TYYMANVSNFSSLVCDDRYSWTWPDFTCENTFEWDYNFCCNAPWYYYDAATFNNSCMGANDCATGKLYWNSTDMQSWDYDLCRNRTYYMANV
SNFSSLVCDDRYSWTWPDFTCENTFEWDYNFCCNAPWYYYDAATFNNSCMGANDCATGKLYWNSTDMQSWDYDLCRNRTYYMANVSNFSSLV
CDDRYSWTWPDFTCENTFEWDYNFCCNAPWYYYDAATFNNSCMGANDCATGKLYWNSTDMQSWDYDLCRNRTYYMANVSNFSSLVCDDRYSW
TWPDFTCENTFEWDYNFCCNAPWYYYDAATFNNSCMGANDCATGKLYWNSTDMQSWDYDLCRNRTYYMANVSNFSSLVCDDRYSWTWPDFTC
ENTFEWDYNFCCNAPWYYYDAATFNNSCMGANDCATGKLYWNSTDMQSWDYDLCRNRTYYMANVSNFSSLVCDDRYSWTWPDFTCENTFEWD
YNFCCNAPWYYYDAATFNNSCMGANDCATGKLYWNSTDMQSWDYDLCRNRTYYMANVSNFSSLVCDDRYSWTWPDFTCENTFEWDYNFCCNA
PWYYYDAATFNNSCMGANDCATGKLYWNSTDMQSWDYDLCRNRTYYMANVSNFSSLVCDDRYSWTWPDFTCENTFEWDYNFCCNAPWYYYDA
ATFNNSCMGANDCATGKLYWNSTDMQSWDYDLCRNRTYYMANVSNFSSLVCDDRYSWTWPDFTCENTFEWDYNFCCNAPWYYYDAATFNNSC
MGANDCATGKLYWNSTDMQSWDYDLCRNRTYYMANVSNFSSLVCDDRYSWTWPDFTCENTFEWDYNFCCNAPWYYYDAATFNNSCMGANDCA
TGKLYWNSTDMQSWDYDLCRNRTYYMANVSNFSSLVCDDRYSWTWPDFTCENTFEWDYNFCCNAPWYYYDAATFNNSCM
>seq_6
MKQTNTSLVRRAALASITDGLLYLDFFDIFLYGSAAALVFNAVFFPGNDPLLATLCTFATFGVGFIARPLGGIVFGHIGDKHGRKSVMMISI
ILMGLSTLAIGLIPSYYAIGILAPVLLILLRIVMGFAVGGEWAGAVLLVSEHSPNQHRGFFSSWIQTTATAGLFLSLVIVALLPTMLGAEQL
DPWGWRTPFWMEGVPLIVGCLIAILLLYSLREEPAFTAMKEAREADGKMSLRSILEEAWKYGRLVIAIIIGMTVSTVVYYIANWFNVSYGAP
ILKTLNLSAGNTGTLTLFSALWLVMTILTPIPAILICEALGPKWLVAIGLGLVACAIYYVILMPFNSMDKILLFELVFTSIMYGFMTLFCLP
ELPFNEQFGYTSESKRILGRCIFGISAALATTTFPMIHTHLFGGVWTAAGILVSAICVVCVLAYPMYRTQNRIGTEIHEENS
>seq_0
MKQDVNTTTAPEAKPITLHQLRFALFAALEGGTMQLMLLGFLAAIAGALLPGDAGLSSIEILLCLGVPSFMLVFCGAIVGTLVSRPLASAYI
DRLGCARMMPIAFWLSGSLVILLIAYMDWRSIFLGRIILGIASATTPQLIGSLSDISPPQCRGTLTLIQISTLCIGSVFGEILALHTAPWVT
FGIAASTALPAHLLDLADSPPCLAASSRSAVSNSERPRTAESCRIFSCPDSLPPFSHSGTRSFPTMPMRCSCTSCMSAWWPCCSWWSRCWTT
WKSMAMTTSSTRTCRTWAWPNCGSAIRCAIPACCSSWPARRAWLRRRPHRHPDRHRHHHLRPHPDPVPHAHDLDPDQGPQPEPYPVDDEEDD
DDFLYELVAEETANGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGN
GNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNGNG
>seq_33
MKLNQYTTPFVRHAVFAALGGLLFGAEINGAVPPLVPLTETLPGVELGTIASLSAAGCFSGGAIGAPAIGSLIDTYGRVPAMLATLIGVSAL
GSWTAESWRSWIVGRIGETGMGILATTTGIVLAMTLYAPIEAEGRGLSTTIQIAITLGIMISYSVGFLVAGAMSFVACSSCHSRAARPRSPP
TTRCWSAWPSASSSRCSSTRCCCTTSRSRCSAWPPCSCWTTWPPCPTCRTSPPGCAGPPTCRSGWAWSPASTCSSCGWSPTNSPSSPSTTSD
DADGEDDDEEEEDVEAEIEVDLETVLDRLDPERLCIPREDFDLDAVERLRRYHPDRPDGIQLMRDIAKKRELEDMKEALLKGNAEQSIHDVI
SNLSEAIRNPVARSIIMFTAPVFQQFGAAALINVYEPAFGMSSNELGGSNNKSGGNENEFDINFTRISNQTNGAIQQLSKLDVAGITLTSSL
FQLGAVVGSIFAIKLNDPIGRFRLLVNNIFGVIANTAIGFASHQDTTAMVMIILAAITRQLTISCSLSLYYASVCDIHPPEKGTGLGALSNL
MSGLGAIGAMGFLPWIARTSD
```

Listado 12: Mejores y peores secuencias generadas por ProGen2-medium - Config. D.

```
>seq_8
MAFADLLEQVGSTGRFQVLHVTLLSIPILMMASHNLLQNFVALVPPHHCSAHTNLSWFQL
SPEETLQISVPLDPAGRPQRCQRYVTPQWHLLNYNGSSAEEGSDGVLQECMDGWSYNLTV
MTSTIISDWDLVCDLRSLKEMAQTVYMGGVLAGALVFGSLSDRFGRRNLLLISYLLMAVS
GTCTAFSSTFSLFCLFRFGCGMALSGLVLNSFSLIVEWIPTRVRTVTGTMTGYCYTVGQL
ILAAVAYFIRDWRWLTLAVSLPFYAFFLFSWWFHESSRWLALKNKPEEAVEVLKSVARFN
GRQEEGEKIDMLQESMKKEMSNVQGSYSVLDLFRTPKMRIMTICLSVVWLSTSFAYYGLA
MDLQKFGVDIYLIQVIFGAVDIPAKVVITVCMSYVGRRPSQSGALVIAGITILINILVPY
DKQTIRTCLAVVGKGCLAASFNCCYLYSGELYPTILRQNGMGWVSMMARVGAMVAPMVLL
TGDHVPWLPGLIYGGAPILSGIAAIFLPETLGMHLPDTVQDVEEIRTHLLLMFPDLVHSM
SKSSASLLLLLTLYVKFLLCCL
>seq_0
MAAGRTEAQDHPPENGFTPLEHPVPRLLPHIDGSVLPSLGGFGKHQKQLVVLTWIPALFI
GFSQFSDYFLLAQPNGTCVQPLANHSNWTAATPPVTVTGGAPALQLKGNGTGEGYGDGSG
MLCACKEWRLELQTGLSQNVVTKWNLVCDSAWKVHIAKFSLLVGSIFGYLVMGVMADWFG
RHPVLILSVLFMLVFGLSVAFSVNVTMFSTLRFFEGFCLAGLALSLYVLRIELCLPAWRF
SMTMVASFLMVGGQLLMPGVAALCRNWPDRDDWQVLQIVIISPFVLMLPYVWIFPESLRW
LLATQHYRRSKAMMLRIAKKNQVDMTTEKGLKVCVSCFLPELEQELHKKPQRTCIVKMMS
TRNLWKNIVVLCVNSLTGYGIHHCFARSMMDPEAQPTALCHADYYTMAGIAVATCLALCP
AVGLMGRRGGLLTFMIITALASLLQLGLLNLIGKYSLLLRDSLNRRFSVAFSIIGMFSSH
AVSTLSIFFCAEITPTVIRGGGLGLVLASAGFGMLTAPIMELHNQKGYFLHHVIFACCTL
LCIICLLLLPEPRGQPLPESLADGESFTRQTLLHPGEQHLLLAKSDGDYSRVHDTPFDSL
GEDFTAYNSL
>seq_20
MEQYDHIGARLDRLPLARFHFRIFGIISYSLLLTGFLSYSGNVVLAKLVSNGWSNNFLNA
AFTSALMFGYFIGSLTGGFIGDYLGRRKAFRMNLLIVGVSATAASFVPNMYWLIFFRCLM
GTGMGALIMVGYASFTEFIPATVRGKWSARLSFVGNWSPMLSAAIGVVVIAFLSWRMMFL
IGGAGMLLAWYLSGKYFIESPRWLAGKGQQSAAERHLRDVEIQIEKEKNITLPAPPGKPG
THSVNEERGSFWLLFKGQMLRRTLVAITVLIAMNISLYTITVWIPTIFVNSGIDVTKSIF
MTAIIMIGAPVGIFIAALIIDRYPRRLFGSLLLVIIALLGYFYSIQTAEWAILSFGLAML
CFLYMYVCFASAVYVPELWPTHLRLRGSGFVNAVGRIVAVFTPYGVAVLLTQYGSVTVFI
VLGAMLVVCALILFFFGIETRKVSLEEISSLS
>seq_29
MTTTTPPRTRAQRLDALPWTRAHSRILGGSGVGWALDAMDVGLISFVIAQLAVVWEADAG
QLGLVASAGFLGMAIGASVGGLVADRIGRRQVFALTLLVYGVATGVSALAMSVGALIALR
FVVGLGLGAELPVASTLVSEFSPARIRGRVIVILESSWAVGWTAAALIGYLVIPASDDGW
RWALALGAVPAVWAIVVRLRLPESVRFLEAKGRHREAERVVRDLEVAAGADPSTDGAVGP
TAEARVADAAAGSETAVAAP
```

REFERENCIAS BIBLIOGRÁFICAS

[AlQuraishi, 2018] AlQuraishi, M. (2018). Protein linguistics - some thoughts on a mysterious universe. https://moalquraishi.wordpress.com/2018/02/15/protein-linguistics/. Consultado el 14 de octubre de 2024.

[Anand *et al.*, 2022] Anand, N., Eguchi, R., Mathews, I. I., Perez, C. P., Derry, A., Altman, R. B., y Huang, P.-S. (2022). Protein sequence design with a learned potential. *Nature Communications*, 13(1):746.

[Bahdanau, 2014] Bahdanau, D. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[Bepler y Berger, 2021] Bepler, T. y Berger, B. (2021). Learning the protein language: Evolution, structure, and function. *Cell Systems*, 12(6):654–669.e3.

[Blum *et al.*, 2024] Blum, M., Andreeva, A., Florentino, L. C., Chuguransky, S. R., Grego, T., Hobbs, E., Pinto, B. L., Orr, A., Paysan-Lafosse, T., Ponamareva, I., Salazar, G. A., Bordin, N., Bork, P., Bridge, A., Colwell, L., Gough, J., Haft, D. H., Letunic, I., Llinares-López, F., Marchler-Bauer, A., Meng-Papaxanthos, L., Mi, H., Natale, D. A., Orengo, C. A., Pandurangan, A. P., Piovesan, D., Rivoire, C., Sigrist, C. J. A., Thanki, N., Thibaud-Nissen, F., Thomas, P. D., Tosatto, S. C. E., Wu, C. H., y Bateman, A. (2024). Interpro: the protein sequence classification resource in 2025. *Nucleic Acids Research*, 53(D1):D444–D456.

[Chen *et al.*, 2016] Chen, T., Xu, B., Zhang, C., y Guestrin, C. (2016). Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*.

[Chornyi, 2024] Chornyi, A. (2024). Understanding temperature, top-k, and top-p sampling in generative models. https://codefinity.com/blog/Understanding-Temperature%2C-Top-k%2C-and-Top-p-Sampling-in-Generative-Models. Consultado el 23 de mayo de 2025.

[Dao, 2023] Dao, T. (2023). Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

[Das Gupta, 2025] Das Gupta, A. (2025). Perplexity for llm evaluation. https://www.geeksforgeeks.org/perplexity-for-llm-evaluation/. Consultado el 21 de mayo de 2025.

[Dauparas *et al.*, 2022] Dauparas, J., Anishchenko, I., Bennett, N., Bai, H., Ragotte, R. J., Milles, L. F., Wicky, B. I. M., Courbet, A., de Haas, R. J., Bethel, N., Leung, P. J. Y., Huddy, T. F., Pellock, S., Tischer, D., Chan, F., Koepnick, B., Nguyen, H., Kang, A., Sankaran, B., Bera, A. K., King, N. P., y Baker, D. (2022). Robust deep learning–based protein sequence design using proteinmpnn. *Science*, 378(6615):49–56.

[Davis *et al.*, 2010] Davis, T. L., Walker, J. R., Campagna-Slater, V., Finerty, Jr, P. J., Paramanathan, R., Bernstein, G., MacKenzie, F., Tempel, W., Ouyang, H., Lee, W. H., Eisenmesser, E. Z., y Dhe-Paganon, S. (2010). Structural and biochemical characterization of the human cyclophilin family of peptidyl-prolyl isomerases. *PLOS Biology*, 8(7):1–16.

[Dryden *et al.*, 2008] Dryden, D. T., Thomson, A. R., y White, J. H. (2008). How much of protein sequence space has been explored by life on earth? *Journal of The Royal Society Interface*, 5(25):953–956.

[Duman Keles *et al.*, 2023] Duman Keles, F., Wijewardena, P. M., y Hegde, C. (2023). On the computational complexity of self-attention. En Agrawal, S. y Orabona, F., editores, *Proceedings of The 34th International Conference on Algorithmic Learning Theory*, volumen 201 de *Proceedings of Machine Learning Research*, pp. 597–619. PMLR.

[Eddy, 2004] Eddy, S. R. (2004). Where did the blosum62 alignment score matrix come from? *Nature Biotechnology*, 22(8):1035–1036.

[Ferruz *et al.*, 2023] Ferruz, N., Heinzinger, M., Akdel, M., Goncearenco, A., Naef, L., y Dallago, C. (2023). From sequence to function through structure: Deep learning for protein design. *Computational and Structural Biotechnology Journal*, 21:238–250.

[Ferruz *et al.*, 2022] Ferruz, N., Schmidt, S., y Höcker, B. (2022). Protgpt2 is a deep unsupervised language model for protein design. *Nature Communications*, 13(1):4348.

[Fournier *et al.*, 2024] Fournier, Q., Vernon, R. M., van der Sloot, A., Schulz, B., Chandar, S., y Langmead, C. J. (2024). Protein language models: Is scaling necessary? *bioRxiv preprint bioRxiv:2024.09.23.614603*.

[Gauthier *et al.*, 2019] Gauthier, J., Vincent, A. T., Charette, S. J., y Derome, N. (2019). A brief history of bioinformatics. *Briefings in bioinformatics*, 20(6):1981–1996.

[Ghafarollahi y Buehler, 2024] Ghafarollahi, A. y Buehler, M. J. (2024). Protagents: Protein discovery via large language model multi-agent collaborations combining physics and machine learning. *arXiv preprint arXiv:2402.04268*.

[Gibbs, 2020] Gibbs, R. A. (2020). The human genome project changed everything. *Nature Reviews Genetics*, 21(10):575–576.

[González-Torres *et al.*, 2007] González-Torres, L., Téllez-Valencia, A., Sampedro, J. G., y Nájera, H. (2007). Las proteínas en la nutrición. *Revista salud pública y nutrición*, 8(2):1–7.

[Guedes *et al.*, 2011] Guedes, R. L. M., Prosdocimi, F., Fernandes, G. R., Moura, L. K., Ribeiro, H. A. L., y Ortega, J. M. (2011). Amino acids biosynthesis and nitrogen assimilation pathways: a great genomic deletion during eukaryotes evolution. *BMC Genomics*, 12 Suppl 4(Suppl 4):S2.

[Guo *et al.*, 2023] Guo, Z., Liu, J., Wang, Y., Chen, M., Wang, D., Xu, D., y Cheng, J. (2023). Diffusion models in bioinformatics: A new wave of deep learning revolution in action. *arXiv preprint arXiv:2302.10907*.

[Han *et al.*, 2024] Han, Z., Gao, C., Liu, J., Zhang, J., y Zhang, S. Q. (2024). Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*.

[Heaven, 2020] Heaven, W. D. (2020). Deepmind's protein-folding ai has solved a 50-year-old grand challenge of biology. https://www.technologyreview.com/2020/11/30/1012712/deepmind-protein-folding-ai-solved-biology-science-drugs-disease/. Consultado el 13 de mayo de 2025.

[Henikoff y Henikoff, 1992] Henikoff, S. y Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919.

[Hrbáň y Hoksza, 2024] Hrbáň, H. y Hoksza, D. (2024). Protein family sequence generation through progen2 fine-tuning. En *2024 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 7037–7039.

[Hu *et al.*, 2022] Hu, Edward J and Shen, Yelong and Wallis, Phillip and Allen-Zhu, Zeyuan and Li, Yuanzhi and Wang, Shean and Wang, Lu and Chen, Weizhu and others (2022). Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

[Ingraham *et al.*, 2023] Ingraham, J. B., Baranov, M., Costello, Z., Barber, K. W., Wang, W., Ismail, A., Frappier, V., Lord, D. M., Ng-Thow-Hing, C., Van Vlack, E. R., Tie, S., Xue, V., Cowles, S. C., Leung, A., Rodrigues, J. V., Morales-Perez, C. L., Ayoub, A. M., Green, R., Puentes, K., Oplinger, F., Panwar, N. V., Obermeyer, F., Root, A. R., Beam, A. L., Poelwijk, F. J., y Grigoryan, G. (2023). Illuminating protein space with a programmable generative model. *Nature*, 623(7989):1070–1078.

[IUPAC, 1968] IUPAC (1968). Iupac-iub commission on biochemical nomenclature a one-letter notation for amino acid sequences. *Journal of Biological Chemistry*, 243(13):3557–3559.

[Jumper *et al.*, 2021] Jumper, John and Evans, Richard and Pritzel, Alexander and Green, Tim and Figurnov, Michael and Ronneberger, Olaf and Tunyasuvunakool, Kathryn and Bates, Russ and Žídek, Augustin and Potapenko, Anna and others (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.

[Kalra y Barkeshli, 2024] Kalra, D. S. y Barkeshli, M. (2024). Why warmup the learning rate? underlying mechanisms and improvements. En Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., y Zhang, C., editores, *Advances in Neural Information Processing Systems*, volumen 37, pp. 111760–111801. Curran Associates, Inc.

[Kendall, 1952] Kendall, M. G. (1952). George udny yule c.b.e., f.r.s. *Journal of the Royal Statistical Society. Series A (General)*, 115(1):156–161.

[Keskar *et al.*, 2019] Keskar, N. S., McCann, B., Varshney, L. R., Xiong, C., y Socher, R. (2019). Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

[Kortemme, 2024] Kortemme, T. (2024). De novo protein design-from new structures to programmable functions. *Cell*, 187(3):526–544.

[Koti, 2024] Koti, V. (2024). From theory to code: Step-by-step implementation and code breakdown of gpt-2 model. https://medium.com/@vipul.koti333/from-theory-to-code-step-by-step-implementation-and-code-breakdown-of-gpt-2-model-7bde8d5cecda. Consultado el 22 de abril de 2025.

[Lee *et al.*, 2025] Lee, J. S., Abdin, O., y Kim, P. M. (2025). Language models for protein design. *Current Opinion in Structural Biology*, 92:103027.

[Lin *et al.*, 2024] Lin, Y., Lee, M., Zhang, Z., y AlQuraishi, M. (2024). Out of many, one: Designing and scaffolding proteins at the scale of the structural universe with genie 2. *arXiv preprint arXiv:2405.15489*.

[Lin *et al.*, 2022] Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., Smetanin, N., Verkuil, R., Kabeli, O., Shmueli, Y., dos Santos Costa, A., Fazel-Zarandi, M., Sercu, T., Candido, S., y Rives, A. (2022). Evolutionary-scale prediction of atomic level protein structure with a language model. *bioRxiv preprint bioRxiv:2022.07.20.500902*.

[Lipman y Pearson, 1985] Lipman, D. J. y Pearson, W. R. (1985). Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441.

[Loshchilov y Hutter, 2017] Loshchilov, I. y Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

[Lv *et al.*, 2024] Lv, L., Lin, Z., Li, H., Liu, Y., Cui, J., Chen, C. Y.-C., Yuan, L., y Tian, Y. (2024). Prollama: A protein large language model for multi-task protein language processing. *arXiv preprint arXiv:2402.16445*.

[Madani *et al.*, 2023] Madani, Ali and Krause, Ben and Greene, Eric R and Subramanian, Subu and Mohr, Benjamin P and Holton, James M and Olmos, Jose Luis and Xiong, Caiming and Sun, Zachary Z and Socher, Richard and others (2023). Large language models generate functional protein sequences across diverse families. *Nature Biotechnology*, 41(8):1099–1106.

[Majumder y Proskurin, 2021] Majumder, R. y Proskurin, A. (2021). Deepspeed: Accelerating large-scale model inference and training via system optimizations and compression. https://www.microsoft.com/en-us/research/blog/deepspeed-accelerating-large-scale-model-inference-and-training-via-system-optimizations-and-compression/. Consultado el 28 de abril de 2025.

[Marchler-Bauer y Bryant, 2004] Marchler-Bauer, A. y Bryant, S. H. (2004). Cd-search: protein domain annotations on the fly. *Nucleic Acids Res*, 32(Web Server issue):W327–31.

[Micikevicius *et al.*, 2017] Micikevicius, Paulius and Narang, Sharan and Alben, Jonah and Diamos, Gregory and Elsen, Erich and Garcia, David and Ginsburg, Boris and Houston,

Michael and Kuchaiev, Oleksii and Venkatesh, Ganesh and others (2017). Mixed precision training. *arXiv preprint arXiv:1710.03740*.

[Milmo, 2023] Milmo, D. (2023). Chatgpt reaches 100 million users two months after launch. https://www.theguardian.com/technology/2023/feb/02/chatgpt-100-million-users-open-ai-fastest-growing-app. Consultado el 19 de abril de 2025.

[Mirdita *et al.*, 2022] Mirdita, M., Schütze, K., Moriwaki, Y., Heo, L., Ovchinnikov, S., y Steinegger, M. (2022). Colabfold: making protein folding accessible to all. *Nature Methods*, 19(6):679–682.

[Naraoka *et al.*, 2023] Naraoka, H., Takano, Y., Dworkin, J. P., Oba, Y., Hamase, K., Furusho, A., Ogawa, N. O., Hashiguchi, M., y Fukushima, K. (2023). Soluble organic molecules in samples of the carbonaceous asteroid (162173) ryugu. *Science*, 379(6634):eabn9033.

[Nelson *et al.*, 2019] Nelson, D. L., Cox, M. M., y Lehninger, A. L. (2019). *Principios de Bioquímica*. Ediciones Omega, Barcelona.

[Nijkamp *et al.*, 2023] Nijkamp, E., Ruffolo, J. A., Weinstein, E. N., Naik, N., y Madani, A. (2023). Progen2: Exploring the boundaries of protein language models. *Cell Systems*, 14(11):968–978.e3.

[Ofer *et al.*, 2021] Ofer, D., Brandes, N., y Linial, M. (2021). The language of proteins: Nlp, machine learning & protein sequences. *Computational and Structural Biotechnology Journal*, 19:1750–1758.

[Ooi *et al.*, 2025] Ooi, K.-B., Tan, G. W.-H., Al-Emran, M., Al-Sharafi, M. A., Capatina, A., Chakraborty, A., Dwivedi, Y. K., Huang, T.-L., Kar, A. K., Lee, V.-H., Loh, X.-M., Micu, A., Mikalef, P., Mogaji, E., Pandey, N., Raman, R., Rana, N. P., Sarker, P., Sharma, A., Teng, C.-I., Wamba, S. F., y Wong, L.-W. (2025). The potential of generative artificial intelligence across disciplines: Perspectives and future directions. *Journal of Computer Information Systems*, 65(1):76–107.

[Pan y Kortemme, 2021] Pan, X. y Kortemme, T. (2021). Recent advances in de novo protein design: Principles, methods, and applications. *Journal of Biological Chemistry*, 296:100558.

[Pettersen *et al.*, 2021] Pettersen, E. F., Goddard, T. D., Huang, C. C., Meng, E. C., Couch, G. S., Croll, T. I., Morris, J. H., y Ferrin, T. E. (2021). UCSF ChimeraX: Structure visualization for researchers, educators, and developers. *Protein Sci.*, 30(1):70–82.

[Prince, 2023] Prince, S. J. (2023). *Understanding Deep Learning*. The MIT Press.

[Radford *et al.*, 2019] Radford, Alec and Wu, Jeffrey and Child, Rewon and Luan, David and Amodei, Dario and Sutskever, Ilya and others (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[Richardson y Richardson, 1989] Richardson, J. S. y Richardson, D. C. (1989). The de novo design of protein structures. *Trends in biochemical sciences*, 14(7):304–309.

[Sandhu, 2024] Sandhu, J. A. (2024). What are llms and generative ai? a beginner's guide to the technology turning heads. https://srinstitute.utoronto.ca/news/gen-ai-llms-explainer. Consultado el 20 de abril de 2025.

[Shafee, 2014] Shafee, T. (2014). *Evolvability of a viral protease: experimental evolution of catalysis, robustness and specificity*. Tesis doctoral, Apollo - University of Cambridge Repository.

[Siddique, 2023] Siddique, Y. (2023). Understanding the controllable parameters to run/inference your large language model. https://medium.com/@developer.yasir.pk/understanding-the-controllable-parameters-to-run-inference-your-large-language-model-30643bb46434. Consultado el 23 de mayo de 2025.

[Smith y Waterman, 1981] Smith, T. y Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197.

[Sohl-Dickstein *et al.*, 2015] Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., y Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*.

[Song *et al.*, 2023] Song, Z., Zhang, Q., Wu, W., Pu, Z., y Yu, H. (2023). Rational design of enzyme activity and enantioselectivity. *Frontiers in Bioengineering and Biotechnology*, 11.

[Taori *et al.*, 2023] Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., y Hashimoto, T. B. (2023). Alpaca: A strong, replicable instruction-following model. https://crfm.stanford.edu/2023/03/13/alpaca.html. Consultado el 21 de abril de 2025.

[Toews, 2021] Toews, R. (2021). Alphafold is the most important achievement in ai—ever. https://www.forbes.com/sites/robtoews/2021/10/03/alphafold-is-the-most-important-achievement-in-ai-ever/. Consultado el 11 de octubre de 2024.

[Toews, 2023] Toews, R. (2023). The next frontier for large language models is biology. https://www.forbes.com/sites/robtoews/2023/07/16/the-next-frontier-for-large-language-models-is-biology/. Consultado el 10 de mayo de 2025.

[Touvron *et al.*, 2023] Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E.,

Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., y Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

[Vaswani *et al.*, 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., y Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Watson *et al.*, 2023] Watson, J. L., Juergens, D., Bennett, N. R., Trippe, B. L., Yim, J., Eisenach, H. E., Ahern, W., Borst, A. J., Ragotte, R. J., Milles, L. F., Wicky, B. I. M., Hanikel, N., Pellock, S. J., Courbet, A., Sheffler, W., Wang, J., Venkatesh, P., Sappington, I., Torres, S. V., Lauko, A., De Bortoli, V., Mathieu, E., Ovchinnikov, S., Barzilay, R., Jaakkola, T. S., DiMaio, F., Baek, M., y Baker, D. (2023). De novo design of protein structure and function with rfdiffusion. *Nature*, 620(7976):1089–1100.

[Woolfson, 2021] Woolfson, D. N. (2021). A brief history of de novo protein design: Minimal, rational, and computational. *Journal of Molecular Biology*, 433(20):167160. From Protein Sequence to Structure at Warp Speed: How Alphafold Impacts Biology.

[Xu y Nussinov, 1998] Xu, D. y Nussinov, R. (1998). Favorable domain size in proteins. *Folding and Design*, 3(1):11–17.

[Yang *et al.*, 2020] Yang, M., Derbyshire, M. K., Yamashita, R. A., y Marchler-Bauer, A. (2020). Ncbi's conserved domain database and tools for protein domain analysis. *Current Protocols in Bioinformatics*, 69(1):e90.

[Zhang *et al.*, 2024] Zhang, S., Dong, L., Li, X., Zhang, S., Sun, X., Wang, S., Li, J., Hu, R., Zhang, T., Wu, F., y Wang, G. (2024). Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*.

[Zhao *et al.*, 2025] Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.-Y., y Wen, J.-R. (2025). A survey of large language models. *arXiv preprint arXiv:2303.18223v16*.

[Zhuo *et al.*, 2024] Zhuo, L., Chi, Z., Xu, M., Huang, H., Zheng, H., He, C., Mao, X.-L., y Zhang, W. (2024). Protllm: An interleaved protein-language llm with protein-as-word pre-training. *arXiv preprint arXiv:2403.07920*.