Joel Lacambra Chen

Edición de drogas para la predicción de la afinidad de unión proteína-fármaco

TRABAJO DE FIN DE GRADO

dirigido por Dr. Francesc Serratosa i Casanelles

Grado de Ingeniería Informática

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2025

Resum.

Aquest Treball de Fi de Grau tracta el desenvolupament d'una eina que combina la visualització i edició molecular amb tècniques d'intel·ligència artificial per a la predicció d'afinitats entre molècules i la proteasa principal del SARS-CoV-2, una tasca fonamental en àmbits com la bioinformàtica i el descobriment de nous fàrmacs. El projecte parteix de la necessitat de disposar d'una solució nativa, simple i personalitzable basada exclusivament en Python, que integri la manipulació molecular mitjançant *RDKit* i el disseny d'interfícies gràfiques amb biblioteques com PySide6.

Per a això, s'ha dissenyat una interfície gràfica que permet a l'usuari visualitzar i editar molècules de manera interactiva. Paral·lelament, s'han implementat diversos models de xarxes neuronals gràfiques (Graph Neural Networks, GNN[1]), com GCN[2], GIN[3] i GINE[4], utilitzant la biblioteca *PyTorch Geometric* i altres tècniques complementàries per modelar, entrenar i predir valors d'afinitat molecular. De manera que tot el que s'havia de fer

Els models s'han avaluat amb mètriques estàndard de regressió per comprovar-ne la precisió. Els resultats obtinguts demostren la viabilitat de l'enfocament proposat, assolint una eina funcional que integra visualització, edició i predicció molecular dins un entorn Python lleuger i flexible.

[1] GNN: Graph Neural Network

[2] GCN: Graph Convolutional Network

[3] GIN: Graph Isomorphism Network

[4] GINE: Graph Isomorphism Network with Edge features

Resumen.

Este Trabajo de Fin de Grado aborda el desarrollo de una herramienta que combina la visualización y edición molecular con técnicas de inteligencia artificial para la predicción de afinidades entre moléculas y la proteasa principal del SARS-CoV-2, una tarea relevante en campos como la bioinformática y el descubrimiento de nuevos fármacos. El proyecto parte de la necesidad de disponer de una solución nativa, simple y personalizable basada exclusivamente en Python, utilizando *RDKit* para la manipulación molecular y PySide6 para el diseño de la interfaz gráfica.

Para ello, se ha desarrollado una interfaz gráfica que permite al usuario visualizar y editar moléculas de forma interactiva. Además, se han implementado varios modelos de redes neuronales gráficas (Graph Neural Networks, GNN), como GCN, GIN y GINE, empleando la biblioteca *PyTorch Geometric* junto con otras técnicas para modelar, entrenar y predecir valores de afinidad molecular.

Los modelos han sido evaluados mediante métricas estándar de regresión con el fin de comprobar su precisión. Los resultados obtenidos demuestran la viabilidad del enfoque propuesto, dando lugar a una herramienta funcional que integra visualización, edición y predicción molecular en un entorno ligero y completamente desarrollado en Python.

Abstract.

This Final Degree Project focuses on the development of a tool that combines molecular visualization and editing with artificial intelligence techniques for predicting the binding affinity between molecules and the main protease of SARS-CoV-2, an important task in fields such as bioinformatics and drug discovery. The project aims to provide a native, simple, and customizable solution developed entirely in Python, using *RDKit* for molecular manipulation and PySide6 for graphical interface design.

A graphical interface was developed to allow users to interactively visualize and edit molecular structures. In parallel, several Graph Neural Network (GNN) models were implemented, including GCN, GIN, and GINE, using the *PyTorch Geometric* library along with various techniques for modeling, training, and predicting molecular affinity values.

The models were evaluated using standard regression metrics to assess their predictive accuracy. The results demonstrate the feasibility of the proposed approach, resulting in a functional tool that integrates molecular visualization, editing, and affinity prediction within a lightweight environment built entirely in Python.

Índice

Índice de tablas

Índice de figuras

1 Introducción

La bioinformática ha experimentado avances revolucionarios en la última década gracias a la IA[5]. Entre las tecnologías que más impacto han tenido, destacan las GNN, una arquitectura que ha transformado significativamente el análisis de datos moleculares y la predicción de interacciones en este ámbito. Las GNN permiten procesar y aprender directamente de datos estructurados en forma de grafos, lo que las hace especialmente útiles para modelar relaciones complejas entre objetos que no pueden representarse fácilmente en estructuras convencionales como matrices o secuencias.

El auge de las GNN comenzó específicamente en 2017, aunque su primera mención fue en un artículo de 2009 titulado "The Graph Neural Network Model" por Franco Scarselli y sus colegas [1], cuando se propusieron arquitecturas fundamentales como la GCN de Kipf y Welling (2016) [2] y la GAT de Veličković et al. (2017) [3], marcando un hito importante en el campo. Estos desarrollos hicieron que las GNN fueran más prácticas, escalables y aplicables a una variedad de dominios, impulsando un aumento exponencial en la actividad investigadora.

Según un estudio bibliométrico [4], las publicaciones relacionadas con las GNN crecieron de manera exponencial, con un aumento del 447% en la producción de investigación anual de 2017 a 2019. Este crecimiento se tradujo en la adopción de GNN por parte de la industria a principios de los 2020, donde grandes empresas tecnológicas como Google, Uber, Alibaba, Twitter y Pinterest, entre otros, comenzaron a implementar soluciones basadas en GNN para productos clave, como los sistemas de recomendación

En este contexto, el uso de las GNN ha permitido avances significativos en la predicción de afinidades moleculares, acelerando la identificación de compuestos con potencial terapéutico.

[5] IA: Inteligencia Artificial

El estudio de los ligandos y su interacción con proteínas es esencial para el desarrollo de nuevos tratamientos. La predicción de la afinidad entre un ligando y una proteína, como la proteasa principal del SARS-CoV-2, una enzima clave en la replicación del virus, puede proporcionar información crucial para el diseño de fármacos más eficaces. Aunque los métodos experimentales para estudiar estas interacciones son valiosos, suelen ser lentos y costosos. En contraste, las GNN, al modelar de forma eficiente las interacciones atómicas, ofrecen una alternativa rápida y generalmente precisa.

Para llevar a cabo el entrenamiento de los modelos GNN, se ha utilizado la base de datos *URV-May-2024*, que recopila inhibidores no covalentes de la proteasa principal del SARS-CoV-2 junto con sus valores de afinidad (IC50). No obstante, cabe destacar que el enfoque propuesto en este trabajo es independiente de esta base de datos concreta y puede aplicarse a cualquier conjunto de datos que incluya estructuras moleculares y valores cuantitativos de actividad biológica.

1.1 Objetivo general

El objetivo principal de este proyecto es el desarrollo de una herramienta interactiva para la edición y análisis de moléculas mediante el uso exclusivo del lenguaje de programación Python, integrando capacidades de visualización y edición molecular, así como la aplicación de modelos de aprendizaje automático, en concreto redes neuronales gráficas, para la predicción de su afinidad con la proteasa principal del virus SARS-CoV-2.

Este proyecto nace no tanto de un interés específico en el ámbito de la química, sino del deseo personal de afrontar un reto técnico que combine varias áreas clave de la ingeniería informática: el desarrollo de interfaces gráficas, la integración de bibliotecas especializadas, y la implementación de modelos de machine learning con arquitecturas modernas.

Desde esta perspectiva, el objetivo general no es solo construir una herramienta funcional, sino también consolidar conocimientos adquiridos durante la carrera, como la programación orientada a objetos, el diseño modular, el trabajo con librerías científicas, y el

entrenamiento de modelos con conjuntos de datos reales, aunque reducidos. Este TFG[6] representa una oportunidad para aplicar de forma práctica lo aprendido, enfrentando un problema complejo desde el punto de vista del desarrollo de software.

A través del desarrollo de este proyecto, busco no solo cumplir con los requisitos académicos del TFG, sino también explorar el proceso completo de construcción de una aplicación técnica: desde la concepción inicial, el diseño e implementación, hasta la evaluación de resultados. En definitiva, se trata de un trabajo centrado en el desarrollo de soluciones informáticas robustas y útiles, más allá del dominio específico en el que se apliquen.

1.2 Objetivos específicos

1. Desarrollar una aplicación gráfica para la edición de estructuras moleculares, utilizando *PySide6* como base tecnológica. Esta interfaz permitirá al usuario crear, visualizar y modificar la estructura interna de una molécula de forma interactiva, facilitando la manipulación de átomos y enlaces, así como la importación/exportación de estructuras en formatos estándar (como .sdf[7]).

2. Integrar la aplicación gráfica con la librería *RDKit*, que es la herramienta principal para el tratamiento y análisis de estructuras químicas. *RDKit* proporciona las funcionalidades necesarias para interpretar, manipular y visualizar moléculas a nivel computacional, permitiendo realizar operaciones como la conversión entre formatos, el cálculo de propiedades moleculares, y la generación de representaciones gráficas, entre otras capacidades esenciales para el funcionamiento de la aplicación.

[6] TFG: Trabajo de Fin de Grado

[7] SDF: Structured-Data File

3. Preprocesar un conjunto de datos molecular en formato *SMILES*[8], que incluye tanto las estructuras como sus afinidades de enlace contra la proteasa principal del SARS-CoV-2, para su uso en el entrenamiento, validación y testeo de los modelos. Este paso incluye la transformación de las moléculas en grafos adecuados para las GNNs, considerando atributos de nodos (átomos) y aristas (enlaces).

4. Implementar y entrenar modelos de GNNs, en concreto los modelos GCN, GIN y GINE adaptados a tareas de regresión para predecir la afinidad molecular (valor continuo) frente a la proteasa del SARS-CoV-2, sin utilizar particularmente esta última.

5. Evaluar el rendimiento de los modelos GNN en el problema de regresión propuesto, comparando su capacidad predictiva mediante métricas estándar como el MSE[9].

6. Analizar el comportamiento de los modelos ante un conjunto de datos reducido, discutiendo las implicaciones del tamaño del dataset, la capacidad de generalización de los modelos y posibles estrategias para mitigar el sobreajuste, como el uso de técnicas de regularización.

[8] SMILES: Simplified Molecular Input Line Entry System

[9] MSE: Mean Squared Error

1.3 Diagrama de GANTT

El desarrollo del proyecto se ha planteado en varias fases diferenciadas, siguiendo una metodología de tipo iterativa y modular que permite avanzar paralelamente en distintas áreas del TFG y facilitar su integración final:

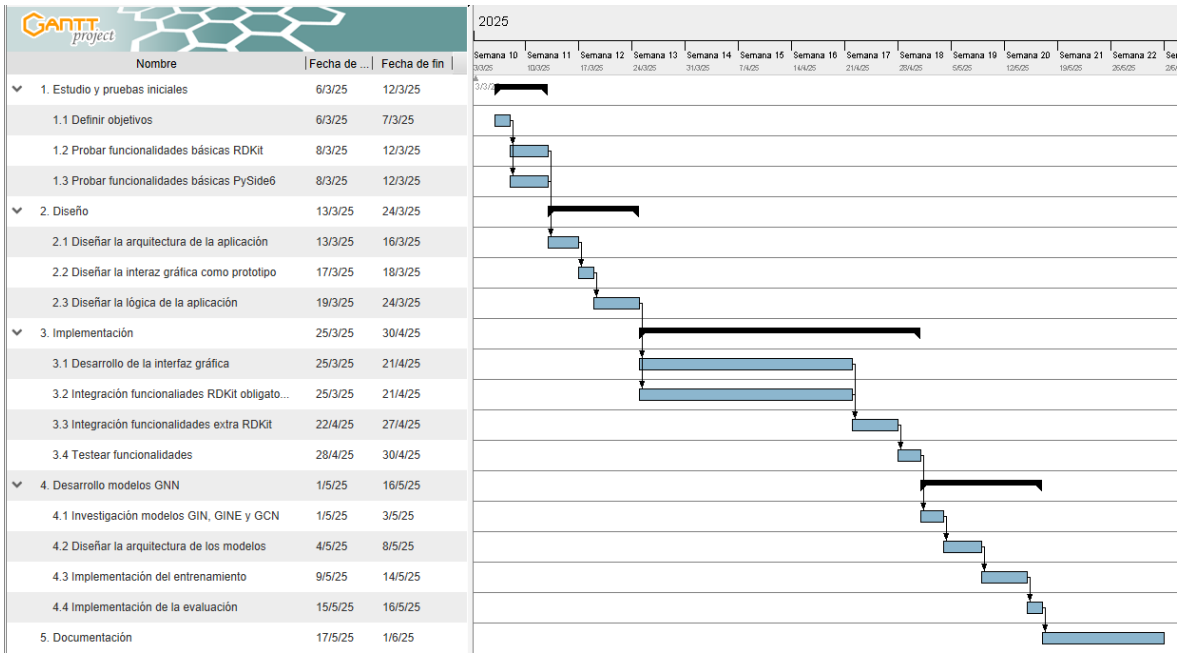

Figura 1. Diagrama de GANTT

1.3.1 Estudio y pruebas iniciales

En una fase inicial de diseño e investigación preliminar, se llevó a cabo un pequeño estudio de las herramientas y tecnologías disponibles para el desarrollo del proyecto. Durante esta etapa se evaluaron distintas opciones para la creación de interfaces gráficas en Python, optando finalmente por PySide6. Una vez finalizada la definición de objetivos de la aplicación, en paralelo, se realizaron pruebas prácticas tanto con la biblioteca Pyside6 y RDKit, para explorar sus capacidades en el manejo y representación de estructuras moleculares y familiarizarme con su sintaxi. Estas pruebas permitieron verificar funcionalidades clave como la lectura y visualización de archivos .sdf, el acceso a propiedades moleculares entre otros.

1.3.2 Diseño

Una vez completada la etapa de investigación, se procedió al diseño de la solución. Esta fase incluyó tanto la planificación de la arquitectura general de la aplicación, como la elaboración de un primer prototipo de interfaz gráfica. Se definió una arquitectura modular que permitiera separar claramente la lógica de negocio, la interfaz y facilitando la escalabilidad y el mantenimiento del sistema. En lo que respecta al diseño de la interfaz gráfica, se construyeron los primeros prototipos del editor molecular, enfocándose en la experiencia del usuario y en una representación visual clara de las moléculas. El prototipo incluía funciones esenciales como la carga de archivos .sdf, la visualización estructural de compuestos químicos, y diversas funcionaliades que se explicarán más adelante.

1.3.3 Implementación

En esta fase se realizó l desarrollo como tal de la aplicación. Se comenzó con la implementación de la interfaz gráfica del usuario, integrando las funcionalidades previamente esbozadas en el diseño. La GUI se construyó como un editor molecular que permitiera importar, visualizar y manipular estructuras químicas. Para ello, de forma paralela, se implementaron funciones que conectaban directamente con las capacidades de RDKit. Se desarrollaron diversas herramientas que mejoraban la experiencia del usuario, como botones, menús contextuales e información detallada al interactuar con diferentes partes de la molécula. Una vez implementadas todas aquellas funcionalidades obligatorias, como puede ser el añadir/eliminar átomos y enalces, se integraron más funciones extra para realzar la capacidad de la aplicación.

1.3.4 Desarrollo de modelos GNN

A continuación, se abordó la parte relacionada con la implementación de modelos de GNN. Esta etapa supuso una primera fase de investigación, donde se buscó información sobre los modelos a implementar, tanto GINE, GIN y GCN. A partir de este estudio, se adquirieron ciertos fundamentos sobre estos modelos y una idea general de cómo debería ser su diseño a través de diferentes librerías de python. Una vez terminada la generación de los modelos, se pasó al entrenamiento utilizando implementando diversas técnicas, y por último, se procedió a la fase de evaluación para determinar su rendimiento en el problema de predicción planteado.

1.3.5 Documentación

Finalmente, se llevó a cabo la documentación completa del proyecto. Se redactó una memoria técnica que recoge el proceso de desarrollo, incluyendo la selección de tecnologías, el diseño de la arquitectura, la implementación de la interfaz, el tratamiento de datos moleculares y el entrenamiento de los modelos GNN entre muchas otras. Se puso especial atención en describir con detalle tanto los aspectos técnicos de la lógica detrás de la interfaz gráfica como la metodología seguida en la fase de de las GNN.

2 Requisitos

Seguidamente se definen los requisitos necesarios para el correcto desarrollo del sistema. Estos se dividen en dos categorías principales: los *requisitos funcionales* que describen las funcionalidades que la herramienta debe ofrecer al usuario, y los *requisitos no funcionales*, que establecen condiciones adicionales relacionadas con la calidad del software, como la eficiencia, la usabilidad entre otros.

2.1 Requisitos funcionales

1. El sistema debe permitir al usuario cargar moléculas desde archivos en formato .sdf.
2. El sistema debe permitir al usuario visualizar la estructura molecular en 2D.
3. El usuario podrá editar la molécula, añadiendo o eliminando átomos y enlaces.
4. El usuario podrá editar el átomo/enlace y cambiar el elemento sin necesidad de eliminarlo y crearlo de nuevo.
5. El sistema debe notificar al usuario si la molécula es válida.
6. El usuario podrá mover libremente los átomos a través del canvas.
7. El usuario podrá modificar la carga formal del átomo seleccionado.
8. El usuario podrá ver las características de los átomos al pasar el ratón por encima.
9. La aplicación debe permitir al usuario seleccionar uno de los modelos implementados (GCN, GIN, GINE).
10. El sistema debe calcular y mostrar la predicción de afinidad entre la molécula cargada y el modelo seleccionado entrenado.
11. El sistema debe permitir guardar las moléculas editadas en formato sdf.
12. El sistema debe permitir guardar la molécula como formato png[10].
13. El sistema debe notificar si el archivo a guardar no es válido.
14. La herramienta debe funcionar de forma local, sin necesidad de conexión a internet.

[10] PNG: Portable Network Graphics

2.2 Diagrama de Casos de Uso

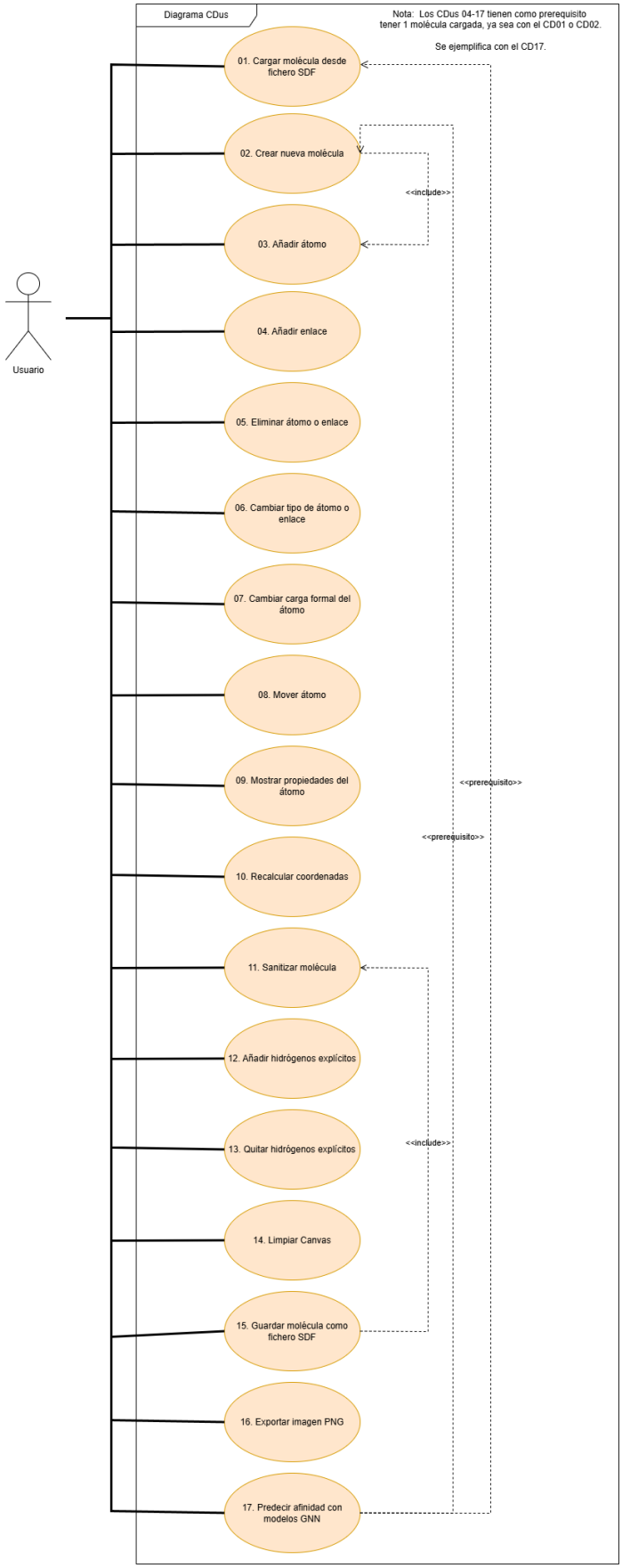

Figura 2. Diagrama de Casos de Uso

Se muestra en la Figura 5 el diagrama de casos de uso específico para el usuario con las diferentes acciones que puede realizar durante el flujo del programa.

2.2.1 Cdu 01. Cargar molécula desde fichero SDF

Resumen de la funcionalidad	Permite cargar una molécula desde un fichero SDF y visualizarla en el canvas.
Parámetros de entrada	Ruta del fichero SDF.
Parámetros de salida	Molécula cargada en el canvas.
Actores	Usuario.
Precondición	Ninguna.
Postcondición	La molécula se visualiza correctamente en el canvas.
Proceso normal principal	1. El usuario hace click u “hover” en el botón de “Archivo”. 2. El usuario pulsa el botón “Cargar SDF”. 3. Selecciona el fichero mediante un diálogo. 4. El sistema lee el fichero y crea la molécula. 5. Se muestra la molécula en el canvas.
Alternativas del proceso y excepciones	Ninguna.

2.2.2 Cdu 02. Crear nueva molécula

Resumen de la funcionalidad	Crear una nueva molécula en un canvas vacío.
Parámetros de entrada	Ninguno.
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	Limpiar el canvas en caso de no cargar un archivo.
Postcondición	El canvas deja de estar vacío por la presencia de al menos 1 átomo.
Proceso normal principal	1. El usuario, en un canvas vacío, ejecuta el caso de uso *Cdu 03. Añadir átomo en un canvas vacío*.
Alternativas del proceso y excepciones	1a. El usuario carga un archivo SDF por lo que la molécula actual se limpia.

2.2.3 Cdu 03. Añadir átomo

Resumen de la funcionalidad	Permite añadir un átomo a la molécula.
Parámetros de entrada	Tipo de átomo a añadir (opcional).
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	Ninguno.
Postcondición	Se añade el átomo a la molécula.
Proceso normal principal	1. El usuario selecciona el botón con label “Añadir átomo”, en la barra de herramientas lateral izquierda. 2. Hace clic en alguna posición del canvas. 3. El sistema añade un átomo (por defecto, carbono).
Alternativas del proceso y excepciones	1a. El usuario puede seleccionar el átomo a utilizar pulsando el botón “Tabla de elementos” antes o después de seleccionar el botón “Añadir átomo”. 2a. La posición es inválida. 2a1. El sistema no añadirá el átomo si está demasiado cerca de otro átomo/enlace. 2b. Hace clic sin haber seleccionado “Añadir átomo”, por lo que no se añadirá ningún átomo

2.2.4 Cdu 04. Añadir enlace

Resumen de la funcionalidad	Permite crear un enlace entre dos átomos.
Parámetros de entrada	Tipo de enlace (opcional).
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	Debe haber al menos dos átomos en el canvas.
Postcondición	Se crea un enlace entre los átomos.
Proceso normal principal	1. El usuario selecciona el botón con label “Añadir enlace”, en la barra de herramientas lateral izquierda. 2. Hace un drag entre dos átomos. 3. El sistema crea un enlace simple por defecto.

Alternativas del proceso y excepciones	1a. El usuario puede seleccionar el enlace a utilizar pulsando el botón “Tipo de enlaces” antes o después de seleccionar el botón “Añadir enlace”. 2a. Ya existe un enlace entre esos átomos, por lo cual se ignora el nuevo enlace. 2b. Si no se selecciona el botón “Tipo de enlaces”, no ocurrirá lo esperado.

2.2.5 Cdu 05. Eliminar átomo o enlace

Resumen de la funcionalidad	Permite eliminar un átomo o un enlace.
Parámetros de entrada	Elemento seleccionado (átomo o enlace).
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	El elemento debe existir.
Postcondición	El elemento se elimina del canvas.
Proceso normal principal	1. El usuario selecciona el botón con label “Eliminar átomo o enlace” en la barra de herramientas lateral izquierda. 2. El usuario selecciona el átomo o enlace. 3. El sistema actualiza la molécula.
Alternativas del proceso y excepciones	2a. Si no existe ningún átomo o enlace, se ignora. 2b. Si no se selecciona el botón “Eliminar átomo o enlace”, no ocurrirá lo esperado.

2.2.6 Cdu 06. Cambiar tipo de átomo o enlace

Resumen de la funcionalidad	Permite modificar el tipo de un átomo o de un enlace.
Parámetros de entrada	Átomo o enlace seleccionado, nuevo tipo.
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	El elemento debe existir.
Postcondición	Se actualiza el tipo del elemento.
Proceso normal principal	1. El usuario selecciona el botón con label “Seleccionar átomo o enlace” en la barra de herramientas lateral izquierda. 2. El usuario selecciona el elemento con el click izquierdo.

	3. Escoge el nuevo tipo desde el menú. 4. El sistema actualiza el átomo o enlace.
Alternativas del proceso y excepciones	2a. Si no se selecciona el botón “Seleccionar átomo o enlace”, no ocurrirá lo esperado.

2.2.7 Cdu 07. Cambiar carga formal del átomo

Resumen de la funcionalidad	Permite modificar la carga formal de un átomo.
Parámetros de entrada	Átomo seleccionado, nueva carga.
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	El átomo debe existir.
Postcondición	La carga del átomo se actualiza.
Proceso normal principal	1. El usuario selecciona el botón con label “Seleccionar átomo o enlace” en la barra de herramientas lateral izquierda. 2. El usuario selecciona un átomo con el click derecho. 3. Se abre un diálogo que permite modificar la carga formal. 4. El usuario cierra el diálogo. 5. El sistema aplica el cambio.
Alternativas del proceso y excepciones	2a. En el caso de no haber seleccionado el botón mencionado en el paso 1, no ocurrirá lo esperado.

2.2.8 Cdu 08. Mover átomo

Resumen de la funcionalidad	Permite arrastrar un átomo a una nueva posición en el canvas.
Parámetros de entrada	Átomo seleccionado, nueva posición.
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	El átomo debe existir en el canvas.
Postcondición	El átomo cambia de posición.
Proceso normal principal	1. El usuario selecciona el botón con label “Seleccionar átomo o enlace” en la barra de herramientas lateral izquierda. 2.El usuario pulsa sobre un átomo y lo arrastra.

	3. El sistema actualiza su posición en tiempo real.
Alternativas del proceso y excepciones	2a. En el caso de no haber seleccionado el botón mencionado en el paso 1, no ocurrirá lo esperado.

2.2.9 Cdu 09. Mostrar propiedades del átomo

Resumen de la funcionalidad	Muestra información del átomo al pasar el ratón por encima.
Parámetros de entrada	Ninguno.
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	Debe haber átomos en el canvas.
Postcondición	Se muestra la información contextual.
Proceso normal principal	1. El usuario pasa el cursor sobre un átomo. 2. El sistema muestra un tooltip con las propiedades.
Alternativas del proceso y excepciones	Ninguna.

2.2.10 Cdu 10. Recalcular coordenadas

Resumen de la funcionalidad	Recalcula las coordenadas 2D de todos los átomos de la molécula para mejorar la disposición visual.
Parámetros de entrada	Ninguno.
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	Debe haber una molécula cargada o creada en el canvas.
Postcondición	Los átomos se reposicionan según un cálculo automático de coordenadas.
Proceso normal principal	1. El usuario pulsa el botón con label “Recalcular coordenadas” en la barra de herramientas lateral izquierda. 2. El sistema ejecuta el algoritmo de disposición 2D. 3. La molécula se redibuja en el canvas con la nueva disposición.
Alternativas del proceso y excepciones	1a. La molécula no tiene estructura válida. 1a1. El sistema muestra un mensaje de error.

2.2.11 Cdu 11. Sanitizar molécula

Resumen de la funcionalidad	Ejecuta el proceso de sanitización sobre la molécula, validando su estructura química.
Parámetros de entrada	Ninguno.
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	Debe haber una molécula en el canvas.
Postcondición	Ninguna.
Proceso normal principal	1. El usuario pulsa el botón “Sanitizar molécula”. 2. El sistema aplica las comprobaciones internas (valencias, cargas, etc.). 3. Se actualiza el estado de la molécula.
Alternativas del proceso y excepciones	2a. La molécula contiene errores químicos. 2a1. El sistema informa al usuario mediante un mensaje de error en la barra de status.

2.2.12 Cdu 12. Añadir hidrógenos explícitos

Resumen de la funcionalidad	Añade todos los átomos de hidrógeno explícitamente en la representación de la molécula.
Parámetros de entrada	Ninguno.
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	La molécula debe estar cargada o creada.
Postcondición	Se añaden los átomos de hidrógeno visibles en el canvas.
Proceso normal principal	1. El usuario pulsa el botón “Añadir H explícitos” en la barra de herramientas horizontal. 2. El sistema calcula y añade los átomos de hidrógeno necesarios. 3. Se actualiza la visualización del canvas.
Alternativas del proceso y excepciones	1a. La molécula ya tiene hidrógenos explícitos. 1a1. El sistema no añade duplicados.

2.2.13 Cdu 13. Quitar hidrógenos explícitos

Resumen de la funcionalidad	Elimina todos los átomos de hidrógeno explícitos de la representación de la molécula.
Parámetros de entrada	Ninguno.
Parámetros de salida	Molécula con hidrógenos eliminados del canvas.
Actores	Usuario.
Precondición	La molécula debe contener hidrógenos explícitos.
Postcondición	Se eliminan los átomos de hidrógeno visibles.
Proceso normal principal	1. El usuario pulsa el botón “Quitar H explícitos” en la barra de herramientas horizontal. 2. El sistema identifica y elimina todos los átomos de hidrógeno visibles. 3. La visualización se actualiza.
Alternativas del proceso y excepciones	1a. No hay hidrógenos explícitos. 1a1. El sistema no realiza ninguna acción. 1b. En caso de que exista un error, el sistema impedirá eliminar los hidrógenos visibles y muestra un mensaje de error en la barra de status.

2.2.14 Cdu 14. Limpiar Canvas

Resumen de la funcionalidad	Elimina la molécula del canvas.
Parámetros de entrada	Molécula actual.
Parámetros de salida	Canvas vacío.
Actores	Usuario.
Precondición	Debe de existir una molécula.
Postcondición	Canvas vacío.
Proceso normal principal	1. El usuario pulsa el botón “Limpiar Canvas” en la barra de herramientas horizontal. 2. El sistema limpia el canvas y elimina la molécula representada. 3. La visualización se actualiza.
Alternativas del proceso y excepciones	Ninguna.

2.2.15 Cdu 15. Guardar molécula como fichero SDF

Resumen de la funcionalidad	Permite guardar la molécula actual en un fichero SDF.
Parámetros de entrada	Ninguno.
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	Debe haber una molécula válida en el canvas.
Postcondición	Se guarda la molécula en el fichero especificado.
Proceso normal principal	1. El usuario hace click u “hover” en el botón de “Archivo”. 2. El usuario pulsa “Guardar como SDF”. 3. El sistema ejecuta el Cdu 11. Sanitizar molécula. 4. El usuario introduce el nombre de la molécula en un diálogo que se abre. 5. Indica la ruta y el nombre del fichero (opcional). 6. El sistema guarda el fichero.
Alternativas del proceso y excepciones	2a. El sistema advierte de que no existe ninguna molécula a guardar por la barra de “status”. 2b. El sistema muestra un diálogo de “warning” si la molécula no pasó la validación. 2b1. Se pregunta si desea guardar una molécula no válida. 2b2. Si el usuario acepta, se vuelve al paso 3. 3a. El usuario no introduce nombre. 3a1. Cancela o cierra diálogo. 3a2. Indica ruta y nombre obligatorios.

2.2.16 Cdu 16. Exportar imagen PNG

Resumen de la funcionalidad	Permite generar una imagen PNG de la molécula actual.
Parámetros de entrada	Ninguno.
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	Debe haber una molécula en el canvas.

Postcondición	Se genera y guarda la imagen.
Proceso normal principal	1. El usuario hace click u “hover” en el botón de “Archivo”. 2. El usuario selecciona “Exportar como PNG”. 3. Indica la ubicación para guardar la imagen. 4. El sistema renderiza la imagen y la guarda.
Alternativas del proceso y excepciones	1a. El sistema advierte de que no existe ninguna molécula a guardar por la barra de “status”.

2.2.17 Cdu 17. Predecir afinidad con modelos GNN

Resumen de la funcionalidad	Ejecuta la predicción de afinidad entre la molécula actual y una proteína usando un modelo GNN.
Parámetros de entrada	Modelo seleccionado (GIN, GINE o GCN).
Parámetros de salida	Ninguno.
Actores	Usuario.
Precondición	La molécula debe ser válida.
Postcondición	Se muestra el valor de afinidad predicho.
Proceso normal principal	1. El usuario hace click u “hover” en el botón de “Predecir afinidad”. 2. Escoge uno de los tres modelos disponibles. 3. El sistema convierte la molécula a un grafo. 4. El modelo calcula la predicción. 5. Se muestra el valor obtenido en un diálogo.
Alternativas del proceso y excepciones	2a. El sistema advierte de que no existe ninguna molécula a guardar por la barra de “status”.

2.3 Requisitos no funcionales

2.3.1 Eficiencia

En cuanto a la eficiencia, el sistema debe ser capaz de procesar las moléculas y realizar las predicciones necesarias de manera rápida. Se espera que la predicción de moléculas de se realice en menos de 2 segundos, garantizando tiempos de respuesta ágiles que no afecten la experiencia del usuario. Además, el sistema debe optimizar el uso de recursos del dispositivo. Durante su ejecución, no debería consumir más del 5% de la CPU ni utilizar más de 500 MB memoria RAM, lo que asegura un rendimiento eficiente incluso en máquinas con recursos limitados.

Es importante mencionar que la primera vez que se ejecuta la aplicación es probable que se demore en abrir. Esto se debe a procesos típicos como la carga inicial de librerías de RDKit (que están escritas en C++ y requieren inicialización de estructuras internas), la compilación a bytecode de los módulos de Python si no están precompilados, y la carga de componentes gráficos de PySide6, como estilos, fuentes y plugins. Todos estos procesos ocurren solo en la primera ejecución, ya que el sistema operativo y Python gestionan posteriormente estas cargas mediante mecanismos de caché, haciendo que las siguientes ejecuciones sean mucho más rápidas.

2.3.2 Usabilidad

La usabilidad es uno de los aspectos más importantes del sistema. La interfaz gráfica debe ser intuitiva y fácil de utilizar, permitiendo que los usuarios puedan cargar, editar y predecir moléculas sin necesidad de conocimientos previos en programación o bioinformática. La experiencia de usuario debe ser fluida y lógica, con elementos visuales que guíen al usuario de manera clara. Además, el sistema debe ofrecer retroalimentación visual inmediata para que el usuario pueda conocer el estado de sus interacciones, como mensajes de confirmación al cargar o guardar una molécula o al finalizar una predicción.

2.3.3 Escalabilidad

La escalabilidad del sistema es otro requisito fundamental. En el futuro, es probable que se necesiten integrar nuevos modelos de redes neuronales gráficas (GNN) o aumentar la

cantidad de datos procesados (moléculas con muchos más átomos). Por ello, el sistema debe ser fácilmente escalable, permitiendo la incorporación de nuevas funcionalidades sin necesidad de reestructurar la arquitectura existente. Además, el sistema debe ser capaz de manejar un volumen creciente de moléculas y realizar predicciones de forma eficiente a medida que aumenten las necesidades del usuario, sin que esto afecte su rendimiento.

2.3.4 Mantenibilidad

La aplicación ha sido diseñada e implementada con un enfoque que facilita futuras modificaciones, ya sea para añadir nuevas funcionalidades o corregir errores en la lógica existente. El código está estructurado de forma clara y modular, lo que permite su comprensión sin grandes dificultades. Además, las funciones y clases cuentan con documentación básica que contribuye a su mantenibilidad, permitiendo que otros desarrolladores puedan trabajar sobre el proyecto sin necesidad de invertir un tiempo excesivo en entender su funcionamiento.

2.3.5 Portabilidad

La aplicación ha sido desarrollada con el objetivo de ser fácilmente portable entre distintos sistemas operativos y entornos de ejecución. Al estar implementada en Python y utilizar bibliotecas multiplataforma como PySide6 y RDKit, puede ejecutarse en Windows, Linux y macOS con cambios mínimos o nulos en el código fuente. Además, las dependencias están gestionadas mediante un archivo de requisitos (requirements.txt), lo que facilita su instalación en nuevos entornos mediante herramientas estándar como pip.

3 Diseño

En este apartado se expondrán y justificarán las decisiones tomadas en relación con el lenguaje de programación, las librerías utilizadas y otros aspectos técnicos relevantes. El objetivo principal del proyecto ha sido desarrollar una interfaz gráfica sencilla pero funcional, que permita demostrar de forma clara y accesible el funcionamiento del programa, garantizando una experiencia de usuario comprensible, fluida y cómoda.

3.1 Arquitectura del sistema

3.1.1 Lenguaje

Para el desarrollo de este proyecto se utilizó el lenguaje de programación Python, tal como se acordó con el tutor del TFG. Más allá de ser una condición establecida, Python es una opción especialmente adecuada para este tipo de proyectos gracias a su sintaxis sencilla, clara y legible, lo que facilita tanto el desarrollo como el mantenimiento del código. Además, es un lenguaje muy versátil y ampliamente utilizado en el ámbito científico y académico, lo que lo convierte en una herramienta ideal para combinar tareas de programación general, análisis de datos, desarrollo de interfaces gráficas y experimentación con modelos de aprendizaje automático.

3.1.2 Entorno de desarrollo

Para el desarrollo del proyecto se ha utilizado el IDE Visual Studio Code, por su ligereza, facilidad de uso y la gran cantidad de extensiones disponibles que facilitan el trabajo con Python. Además, se ha optado por trabajar dentro de un entorno virtual (venv) de Python, donde se han instalado únicamente las dependencias necesarias para el proyecto. Esta práctica permite mantener el sistema limpio, evitar conflictos con otras instalaciones globales y asegurar que el entorno de ejecución sea reproducible en cualquier otra máquina, utilizando un fichero txt con las dependencias requeridas para su funcionamiento completo.

3.1.3 Librerías utilizadas

Para el desarrollo del proyecto se han empleado diversas librerías de Python que permiten abordar de forma modular y eficiente tareas relacionadas con la visualización y edición molecular, el desarrollo de interfaces gráficas para la manipulación directa y la implementación de modelos de aprendizaje automático. A continuación, se describen brevemente las más relevantes:

- *RDKit* [5]: Librería especializada en química computacional. Permite manipular estructuras moleculares, generar coordenadas 2D/3D, obtener información química (átomos, enlaces, propiedades) y convertir entre formatos como SMILES y SDF. Ha sido clave para el procesamiento y visualización de las moléculas.
- *PySide6* [6**]**: Toolkit oficial de Qt para Python. Se ha utilizado para construir la interfaz gráfica del programa, permitiendo una experiencia visual fluida, moderna y multiplataforma.
- *PyTorch* [7]: Framework de machine learning ampliamente utilizado en investigación y producción. En este proyecto ha servido como base para construir y entrenar modelos de redes neuronales capaces de predecir la afinidad molecular.
- *PyTorch Geometric* [8]: Extensión de PyTorch orientada a trabajar con grafos. Resulta fundamental para tratar moléculas como grafos atómicos y aplicar modelos de redes neuronales sobre estructuras no euclidianas.
- *scikit-learn* [9]: Biblioteca muy utilizada para tareas de aprendizaje automático y análisis de datos. En este caso, se ha empleado principalmente para la normalización de valores de entrada y salida (afinidad).

3.2 Descripción de módulos

A continuación, se presenta la estructura de los directorios y subdirectorios generada con “File Tree to Text Generator” [10] junto con respectivos archivos, indicando brevemente el propósito que tiene cada uno:

```
editor-quimico/
│
├── chem/                              # Funciones relacionadas con operaciones químicas
│   ├ draw_utils.py                    # Funciones de visualización molecular (e.g. renderizado de estructuras)
│   └ mol_operations.py                # Funciones para manipular o analizar moléculas
│
├── data/                              # Datos químicos
│   ├ sdf/                             # Moléculas en formato SDF (ligandos) proporcionados
│   │ ├ 5RFA_ligand.sdf                # Molécula del ligando 5RFA
│   │ ├ 5RGV_ligand.sdf                # Molécula del ligando 5RGV
│   │ ├ 5RH2_ligand.sdf                # Molécula del ligando 5RH2
│   │ └ 5RHD_ligand.sdf                # Molécula del ligando 5RHD
│   ├ BondTypes.py                     # Definición de tipos de enlaces químicos
│   └ ptable.py                        # Información de la tabla periódica
│
├── predictor/                         # Módulo de predicción basado en GNNs
│   ├ models/                          # Modelos entrenados (formato .pth)
│   │ ├ GCN_F.pth                      # Modelo entrenado con arquitectura GCN
│   │ ├ GINE_F.pth                     # Modelo entrenado con arquitectura GINE
│   │ └ GIN_F.pth                      # Modelo entrenado con arquitectura GIN
│   ├ scaler/                          # Escaladores para normalizar datos antes de inferencia
│   │ └ scaler_basic.pkl               # Objeto de escalado básico
│   ├ Graph.py                         # Conversión de moléculas en grafos (estructura de entrada para GNN)
│   ├ Model.py                         # Definición de arquitecturas de redes neuronales
│   ├ SimplePred.py                    # Módulo para realizar inferencias con modelos cargados
│   └ TrainingBasic.py                 # Módulo para entrenar modelo
│
├── ui/                                # Interfaz gráfica de usuario (GUI) con PySide6
│   ├ Components/                      # Widgets y elementos reutilizables
│   │ ├ HooverToolButton.py            # Botón personalizado con efecto hover
│   │ └ ToolBar.py                     # Barra de herramientas principal
│   ├ Dialogs/                         # Ventanas emergentes o diálogos
│   │ └ ElementsSelectorDialog.py      # Diálogo para seleccionarelementos químicos/enlaces y visualizar información
│   ├ MainWindow.py                    # Ventana principal de la aplicación
│   └ MolViewer.py                     # Visualización e interacción con moléculas en la GUI con RDKit
│
├── main.py                            # Punto de entrada principal del programa
├── exit.py                            # Lógica de salida y limpieza del programa
├── requirements.txt                   # Lista de dependencias del entorno (para pip)
└── README.md                          # Documentación general del proyecto
```

Figura 3. Representación estructurada del proyecto

En la Figura 3 podemos observar la jerarquía y organización dividida en varios subdirectorios. El proyecto se ha intentado dividir en carpetas principales como *chem/*, que contiene módulos para las operaciones moleculares y visualización; *data/* contiene tanto información sobre los ligandos como diferentes moléculas para poder cargar en la aplicación; */predictor* donde se incluye todo lo necesario para poder hacer predicciones sobre ligandos utilizando las GNNs, y *ui/* se encarga la interfaz gráfica.

Y por último, el archivo *main.py*, que es el “trigger” y actúa como el punto de entrada a la aplicación.

3.3 Diseño inicial de la interfaz gráfica

Para la construcción de la interfaz gráfica, como se ha mencionado anteriormente, se ha utilizado la librería PySide6 del framework de Qt 6. Esta decisión viene principalmente por el simple hecho de que tiene soporte nativo para representar SVGs[11], a diferencia de librerías de creación de interfaces más simples como "Tkinter", que viene instalada de forma predeterminada en python.

Se ha intentando separar la lógica de la interfaz de la lógica de la manipulación molecular, aunque al final es prácticamente imposible aislar una de la otra completamente.

A continuación se intenta describir los elementos más relevantes sin entrar mucho en detalle técnico:

1. Área principal de visualización

Podríamos describir de forma general la aplicación en dos grandes componentes *QmainWindow* y *QsvgWidget*:

- *QMainWindow*: Es el contenedor principal de la interfaz. Permite establecer una zona central donde se carga el canvas de dibujo molecular. Además, organiza las barras laterales y superiores, encapsulando toda la estructura de la aplicación.
- *QsvgWidget*: Este widget será utilizado para renderizar estructuras moleculares en formato SVG, generadas con *RDKit*. Esta elección permite:
    - Dibujar moléculas en alta resolución sin pixelación al hacer zoom.
    - Integrar directamente la salida SVG de la función de la librería *RDKit*.
    - Posibilidad de usar eventos del ratón para permitir la edición directa: selección, arrastre, eliminación o modificación de átomos y enlaces.

[11] SVG: Scalable Vector Graphics

2. “ToolBar” principal (superior)

La barra o “ToolBar” superior contiene menús desplegables y acciones relacionadas con el ciclo de vida de la molécula, con algunas funcionalidades generales pero no tan relevantes como para poner en la barra lateral izquierda. Se han definido los siguientes subcomponentes:

- *Menú “Archivo”*: Despliega un menú contextual con acciones para cargar y guardar archivos SDF, exportar imágenes PNG y salir de la aplicación.
- *Menú “Tabla de elementos”*: Ofrece al usuario la capacidad de seleccionar elementos químicos de la tabla periódica para después añadir nuevos átomos al canvas.
- *Menú “Tipo de enlaces”*: Igual que la tabla anterir, ofrece al usuario la capacidad de seleccionar diversos tipos de enlaces para unir átomos.
- *Acciones directas*: Se incluyen varios botones de acción rápida como “Sanitizar molécula”, "Quitar/Añadir Hidrógenos explícitos", "Limpiar Canvas" y por último y no menos importante, “Predecir Afinidad” con tres modelos; GINE, GIN y GCN.

3. “ToolBar” lateral izquierda

La barra o “ToolBar” lateral izquierda está más orientada a las acciones de edición directas sobre la molécula, motivo por el cual se ha decidido separar del resto y además por no sobrecargar la barra principal. Se ha intentando definifir gráficamente su comportamiento, sustituyendo su acción descriptiva por un “emoji” que representa su intención de forma más intuitiva. A continuación, se especifican dichos botones:

- 🖱️ *Seleccionar átomo o enlace*: Activa el modo de selección.
- ➕ *Añadir átomo*: Añade un nuevo átomo (por defecto, carbono).
- ❌ *Eliminar átomo o enlace*: Permite eliminar elementos seleccionados.
- 🔗 *Añadir enlace*: Crea enlaces entre átomos seleccionados.
- 🔄 *Recalcular coordenadas*: Vuelve a generar las coordenadas 2D para mejorar la visualización.

3.3.1 Prototipo visual de la interfaz

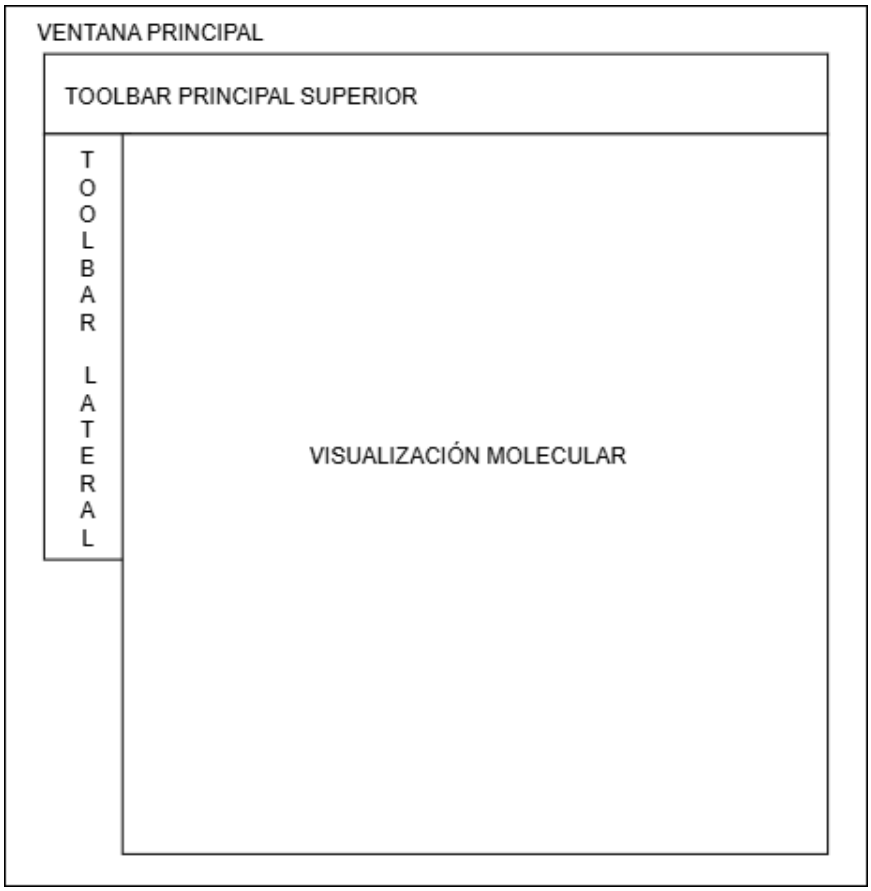

Figura 4. Representación del diseño gáfico

En la Figura 4 podemos observar el diseño descrito anteriormente, resumido en estos 4 grandes componentes:

- Ventana principal que actua como contenedor global para toda la aplicación.
- Toolbar principal superior que contiene funcionalades generales y especificas menos relevantes.
- Toolbar lateral izquierda con acciones directamente relacionadas con la edición y manipulación de la molécula, facilitando el acceso rápido a las herramientas principales.

Visualización molecular muestra la representación gráfica de la molécula en formato SVG, lo cual garantiza su escalabilidad.

3.4 Diseño final de la interfaz gráfica

A continuación, se presentará el diseño final de la interfaz junto con sus diferentes funcionalidades y ciclo de ejecución del programa.

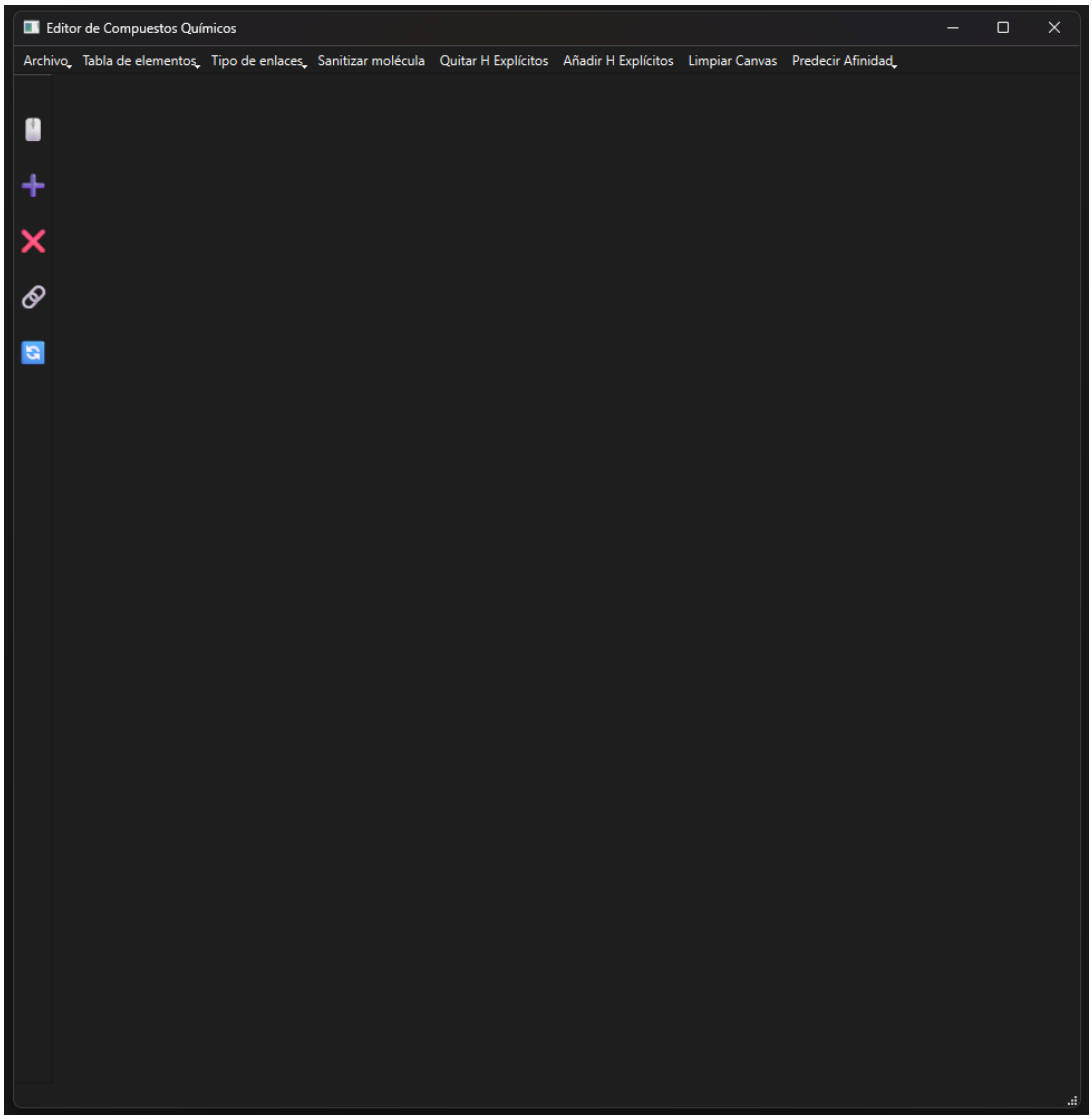

Figura 5. Pantalla inicial

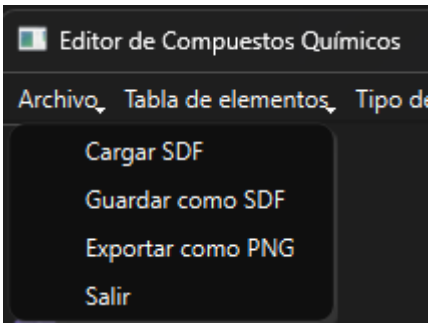

Figura 6. Representación del botón “Archivo”

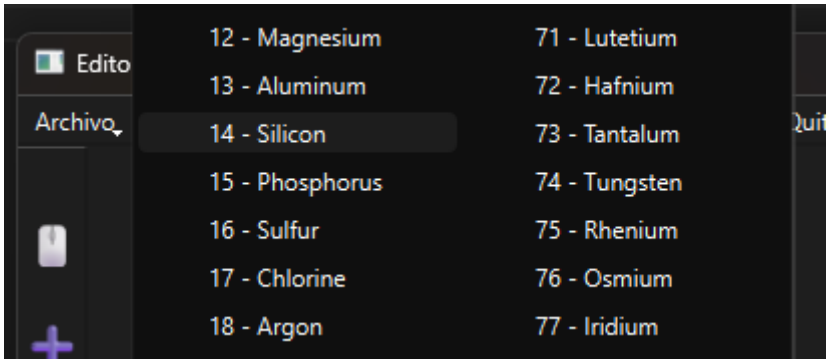

Figura 7. Muestra del botón "Tabla de elementos"

Figura 8. Muestra del botón "Tipo de enlaces"

Figura 9. Representación del botón "Predecir Afinidad"

En la Figura 5 se puede apreciar que se ha seguido fielmente el diseño del prototipo presentando anteriormentete. Además, en las siguientes figuras se muestran las funcionalidades principales de la barra de tareas superior: la Figura 6 presenta las funciones básicas relacionadas con la gestión de archivos; la Figura 7 muestra la tabla con los

elementos disponibles para que el usuario los seleccione; en la Figura 8 se observa la lista de enlaces a elegir; y finalmente, la Figura 9 representan los diferentes modelos que pueden utilizarse para realizar la predicción.

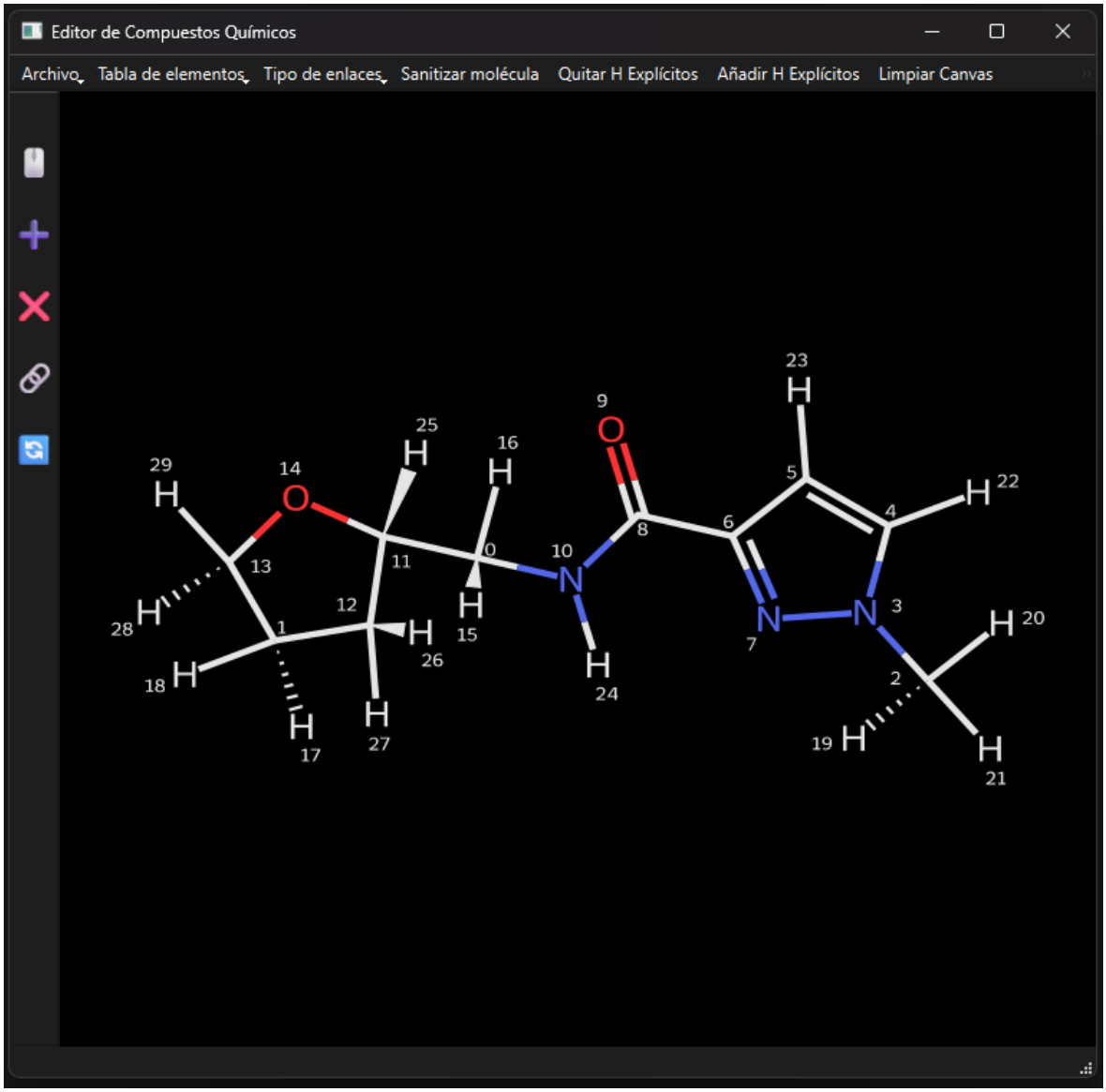

Figura 10. Programa cargado con ligando 5RFA

En la Figura 10 se muestra el programa con el archivo SDF correspondiente al 5RFA ya cargado. Internamente, el sistema se encarga de leer y procesar el archivo, generando una representación visual de la molécula en formato SVG en la pantalla.

Además, se almacenan las coordenadas 2D de la imagen, lo que permite mapear individualmente cada átomo y enlace, facilitando así su posterior manipulación.

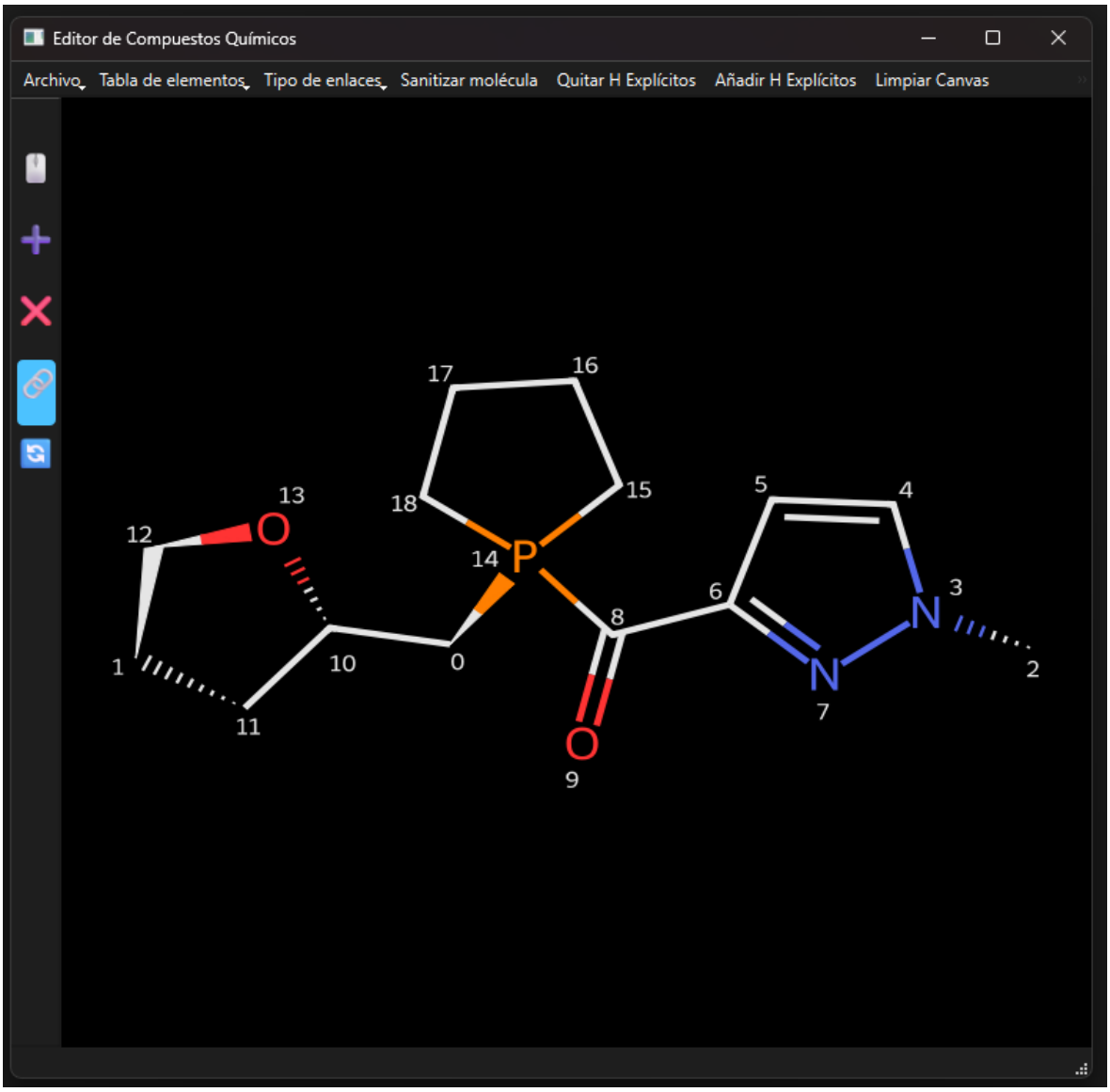

Figura 11. Programa con ligando cargado modificado

En la Figura 11 se muestra el ligando que fue cargado previamente, pero que ha sido modificado de manera ligera. En esta ocasión, se aplicó la función destinada a eliminar los hidrógenos explícitos, simplificando así la estructura visible. Además, se realizó una modificación más específica: se eliminó el átomo con índice 10, correspondiente a un nitrógeno, y en su lugar se insertó un átomo con índice 14, que es un fósforo. Esta sustitución permitió la formación de un anillo compuesto por átomos de carbono, alterando la estructura original de la molécula. Todas estas modificaciones se llevaron a cabo utilizando las funciones previamente descritas para borrar y añadir átomos y enlaces, demostrando la

flexibilidad y capacidad del programa para manipular estructuras químicas de manera precisa y controlada.

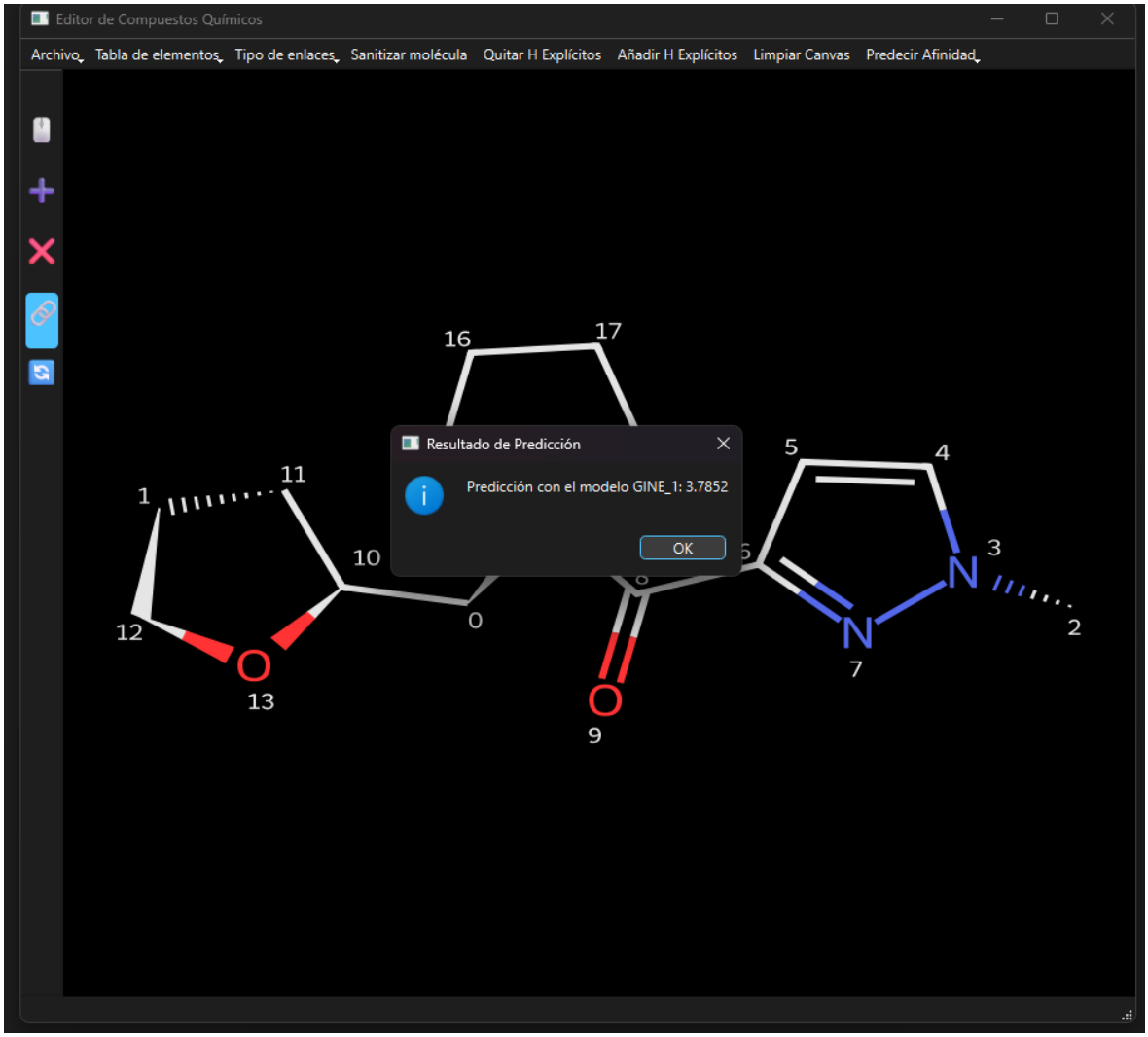

Figura 12. Predición de Afinindad del ligando modificado

En la Figura 12 se puede observar la integración del modelo GNN GINE con la interfaz gráfica, el cual ha sido utilizado para predecir la afinidad de la molécula representada en el canvas. Estos modelos de GNN tienen como objetivo evaluar de manera precisa las interacciones y propiedades moleculares, generando un valor predictivo que refleja las características fundamentales de los átomos y enlaces que componen la estructura molecular.

4 Implementación y uso de algoritmos – Editor Molecular

En esta sección, mi intención es abordar el uso detallado de las bibliotecas que se mencionaron anteriormente en el apartado *3.1.3*, explicando cómo cada una contribuye al desarrollo y funcionamiento de la aplicación. Además, se hará énfasis en algunas de las funciones más importantes que fueron vitales para poder integrar de manera efectiva la parte gráfica con la lógica de manipulación de moléculas. Estas funciones son esenciales para permitir la interacción fluida entre la visualización de las moléculas en el canvas y las operaciones de modificación y análisis que se realizan sobre ellas.

Una vez descrito el uso de las bibliotecas y funciones, se explicará cómo se ha logrado la integración entre la interfaz gráfica de usuario (GUI) y la lógica que maneja la manipulación de las moléculas. Esto incluye la creación de la visualización en 2D de las moléculas y su interacción con el modelo de predicción. El objetivo aquí es mostrar cómo estas dos partes, aparentemente separadas, se complementan para ofrecer una experiencia interactiva y eficiente al usuario, permitiéndole tanto ver como modificar las estructuras moleculares.

Antes de abordar este extenso tema, me gustaría explicar brevemente que son los archivos con extensión ".sdf" que se pueden observar en la Figura 3. Estos archivos, proporcionados específicamente para este proyecto, son de forma general ficheros estructurados que almacenan bases de datos es un formato concreto y estandarizado, permitiendo el intercambio de datos entre diferentes programas sin tener problemas de compatibilidad.

Este tipo de fichero es muy importante y ampliamente usado específicamente en el ámbito de la química computacional y la bioinformática, que permite almacenar estructuras moleculares y datos asociados de manera estructurada, lo que lo convierte en una herramienta muy valiosa para la visualización y el procesamiento de compuestos químicos [11].

4.1 Uso de PySide6

PySide6 es la interfaz de bindings para Qt6, una de las bibliotecas más populares para el desarrollo de aplicaciones gráficas. PySide6 permite a los programadores crear interfaces ricas, interactivas y modernas sin la necesidad de escribir en lenguajes de bajo nivel como C++. Su uso es ideal para el desarrollo de aplicaciones científicas donde se requiere la visualización de datos y la interacción con el usuario.

Algunas de las características más relevantes de PySide6 en este contexto incluyen:

- Widgets gráficos como botones, barras de herramientas y paneles de dibujo para visualización interactiva.
- Gestión de eventos nativa como el clic y el arrastre para interactuar con las moléculas.

4.1.1 Widget de visualización: Integración con QsvgWidget

La visualización de las moléculas se realiza mediante el widget *QSvgWidget* de PySide6, el cual permite cargar y mostrar gráficos en formato SVG. Este formato es ideal para representar moléculas debido a su capacidad para mantener la calidad y escalabilidad sin pérdida de resolución.

La molécula es representada como una imagen SVG generada por RDKit, que contiene los átomos y enlaces de la molécula, así como las coordenadas 2D correspondientes que se utilizan para mapear cada uno de los elementos. El *QSvgWidget* se utiliza para mostrar dicha imagen de manera interactiva. Además, este widget permite la manipulación de la representación de la molécula a medida que el usuario realiza cambios en la estructura (como mover átomos, añadir enlaces, o eliminar átomos) gracias a la posibilidad de utilizar las coordenadas resultantes del ráton del usuario.

4.1.2 Gestión de eventos: Interación con el usuario

Uno de los aspectos clave del proyecto es la interacción del usuario con la estructura molecular, y PySide6 facilita este proceso gracias a su robusto sistema de gestión de eventos.

El sistema de eventos de Qt6 es un componente fundamental que permite manejar de forma eficiente la interacción entre el usuario y la aplicación. Qt utiliza un modelo basado en eventos y señales/slots, que permite capturar acciones del usuario (como clics, arrastres

o movimientos del ratón) y coordinar la comunicación entre los distintos elementos de la interfaz gráfica [12].

4.1.2.1 Eventos (Events)

Los eventos son mensajes que describen una acción que ha ocurrido, como un clic del ratón, el movimiento del ratón, la pulsación de una tecla, etc. Estos eventos son enviados al sistema de eventos de Qt y luego entregados a los objetos que puedan estar interesados en ellos.

4.1.2.2 Objetos de eventos (QEvent)

Los eventos en Qt son representados por instancias de la clase *QEvent* o sus subclases (como QMouseEvent, QKeyEvent, QResizeEvent, etc.). Cada tipo de evento tiene su propia clase, que hereda de QEvent y contiene información específica sobre el evento (como la posición del ratón, qué tecla se presionó, etc.). En mi caso, tenemos el siguiente:

- QMouseEvent: Es la clase utilizada para representar eventos de ratón, como clics y movimientos. En los métodos "handlers" que después describiremos, *mousePressEvent*, *mouseMoveEvent* y *mouseReleaseEvent*, el argumento event es una instancia de *QMouseEvent*.
    - En el caso de mousePressEvent, por ejemplo, se accede a la posición del ratón usando *event.pos()*.

4.1.2.3 Manejo de eventos (Event Handlers)

Los "event handlers" son métodos que se sobrescriben para manejar eventos específicos. En mi caso, se sobrescriben los siguientes manejadores de eventos:

- *mousePressEvent*: Este evento se activa cuando el usuario presiona un botón del ratón (izquierdo o derecho). Dependiendo del modo ("select", "add_atom", "add_bond"), el evento puede realizar diversas acciones, como añadir un átomo o mostrar el inicio de una línea temporal para un enlace.

```
def mousePressEvent(self, event: QtGui.QMouseEvent):
    if event.button() == QtCore.Qt.LeftButton:
        self.start_point = event.pos()
        self.is_dragging = True
     . . .
```

Código 1: Evento generado cuando se hace click con el ratón

- *mouseMoveEvent*: Se activa cuando el ratón se mueve sobre el área de visualización. Aquí se realiza el manejo de la interacción del usuario al mover átomos o al dibujar una línea temporal para un enlace.

```
def mouseMoveEvent(self, event: QtGui.QMouseEvent):
    if self.is_dragging:
        if self.mode == "add_bond":
            self.temp_line_end = event.pos()
        elif self.mode == "select":
     . . .
```

Código 2: Evento generado cuando se mueve el rato

- *mouseReleaseEvent*: Este método se activa cuando el usuario suelta el botón del ratón. Se utiliza para completar las acciones iniciadas en *mousePressEvent* y se realizan comprobaciones para saber si se trata de un clic o un arrastre.

```
def mouseReleaseEvent(self, event: QtGui.QMouseEvent):
    if event.button() == QtCore.Qt.LeftButton:
        self.end_point = event.pos()
        self.is_dragging = False
    . . .
```

Código 3: Evento generado cuando se suela el click del ratón

- *paintEvent*: Este evento se activa cuando es necesario redibujar el widget y ser llama a la función *update()* desde alguna parte del código. En mi caso, se utiliza para dibujar la línea guía de enlace entre átomos.

```
def paintEvent(self, event):
    super().paintEvent(event)  # Llamada a la clase base
    # Dibuja la línea temporal si está en el modo de arrastre
    . . .
```

Código 4: Evento generado cuando se llama a update()

4.1.2.4 Ciclo de eventos (Event Loop)

Aunque en el código del proyecto no se muestra explícitamente el ciclo de eventos, Qt maneja internamente este ciclo cuando ejecutas una aplicación Qt. El ciclo de eventos espera y distribuye los eventos a los widgets correspondientes. Al crear la aplicación Qt y ejecutarla, el ciclo de eventos se pone en marcha y maneja la distribución de eventos a los widgets.

4.1.2.5 Señales y Slots

Qt utiliza un sistema de señales y slots para la comunicación entre objetos. En este proyecto, se utiliza la señal *mol_is_empty* para notificar que la molécula está vacía después

de eliminar átomos o enlaces. Esta señal puede ser conectada a otras partes de la aplicación que deseen reaccionar a este event, en este caso con el código principal de la ventana de la intefaz.

```
. . .
mol_is_empty = QtCore.Signal()
. . .
```

Código 5: Creación de señal para saber cuando el mol no tiene elementos

4.1.3 Actualización de la interfaz: Sincronización entre la visualización y la lógica

A medida que el usuario interactúa con la visualización, es esencial que la interfaz gráfica y la estructura interna de la molécula estén sincronizadas. Esto significa que, cuando, por una parte, se quiere añadir un enlace entre dos átomos, la representación visual debe actualizarse automáticamente para reflejar los cambios y poder observar la línea guía que se utiliza para unir 2 átomos. PySide6 se encarga de la gestión de la actualización de la interfaz mediante el método *update()* del widget, que se mencióno en el apartado 4.1.2.3, lo cual asegura que las modificaciones realizadas por el usuario se reflejen en tiempo real.

Por otra parte, cada vez que se realiza una modificación, como agregar/eliminar un átomo/enlace, la estructura de la molécula se recalcula utilizando diversas funciones con las herramientas de RDKit, que resulta en un nuevo SVG con las modificaciones pertinentes hechas. Este proceso específico se explicará más adelante cuando se entre en del detalle de la integración de las funciones de la librería RDKit.

4.1.4 Personalización de widgets y creación de componentes específicos

En una aplicación gráfica, la personalización de los widgets y la creación de componentes específicos es esencial para proporcionar una experiencia de usuario coherente, funcional y visualmente atractiva. En este proyecto, se han implementado algunos widgets y herramientas personalizadas que mejoran la interacción del usuario con la estructura molecular. Estos componentes incluyen botones interactivos con menús, barras de herramientas especializadas y botones con efectos visuales de hover, todo diseñado para facilitar la manipulación de la molécula de manera eficiente.

A continuación se describen algunos de los componentes clave implementados:

HoverToolButton: El widget HoverToolButton es una extensión personalizada de *QToolButton* que agrega un comportamiento visual especial cuando el cursor del ratón

interactúa con el botón. Este botón es ideal para ser utilizado en las barras de herramientas, ya que muestra un menú desplegable que permite al usuario seleccionar diversas opciones de manera intuitiva, sin tener que hacer click directamente en el botón, lo que ayuda a tener una experiencia más fluida.

Algunas de sus características clave son**:**

- Menú emergente: Al pasar el ratón por encima del botón, se muestra un menú contextual que contiene varias opciones. Esto es útil para agrupar herramientas relacionadas, como las opciones de archivo, átomos o enlaces.
- Desaparece tras salir: Después de que el ratón sale del botón, el menú se cierra automáticamente gracias a un temporizador que gestiona el evento de salida.

Toolbar: La clase ToolBar extiende QToolBar y agrupa una serie de botones y acciones personalizadas que permiten al usuario interactuar con el sistema de moléculas. Esta barra de herramientas utiliza botones con menús emergentes para ofrecer funcionalidades como cargar o guardar archivos, añadir átomos y enlaces, y predecir afinidades moleculares.

LeftToolBar: La clase LeftToolBar es una barra de herramientas vertical que contiene botones de acción que permiten al usuario realizar tareas específicas sobre las moléculas, como seleccionar átomos, añadir enlaces, sanitizar moléculas, y más. Además de las acciones estándar, la barra presenta botones con emojis para mejorar la interacción del usuario, dándole un toque visualmente atractivo y facilitando la accesibilidad.

Algunas de sus características clave son**:**

- Botones con iconos de emojis: Los botones para seleccionar átomos, añadir enlaces, o computar coordenadas están representados con emojis, lo que hace la interfaz más divertida y fácil de entender.
- Acciones de grupo: Se utiliza un *QActionGroup* para asegurar que solo una acción se puede seleccionar a la vez.

4.2 Uso de RDKit

RDKit es una de las bibliotecas más poderosas en el ámbito de la química computacional, utilizada para el manejo, análisis y visualización de estructuras moleculares. En este proyecto, RDKit desempeña un papel crucial en la representación, manipulación y modificación de las moléculas en 2D, proporcionando las funciones necesarias para calcular coordenadas de átomos, generar estructuras moleculares, y gestionar la adición y eliminación de átomos y enlaces. A continuación, se detalla el uso de RDKit en la aplicación interactiva de visualización y edición de moléculas.

4.2.1 Creación y representación de moléculas

Uno de los primeros pasos en cualquier análisis químico es crear una representación de la molécula. RDKit ofrece diversas funciones que permiten la creación de objetos moleculares a partir de representaciones estándar, como archivos en formato .sdf, SMILES o .mol2.

En lugar de usar la cadena SMILES, RDKit permite cargar moléculas desde archivos en formatos comunes como .sdf mediante el uso de la función *Chem.MolFromMolFile()*.

En nuestro caso se hace de la siguiente manera:

```
...
# Cargar la molécula desde un archivo SDF o crear una nueva
        if file_name is not None:
            self.mol = Chem.MolFromMolFile(str(file_name), sanitize=False,
strictParsing=False)
        else:
            self.mol = Chem.RWMol()
...
```

Código 6: Cargar molécula desde archivo SDF o mol vacío

Este código crea un objeto Mol a partir de la estructura molecular almacenada en el archivo .sdf, que se puede manipular igual que cualquier otra molécula cargada mediante SMILES.

Una vez que la molécula es cargada desde el archivo, es común generar una representación 2D para su visualización. RDKit utiliza el módulo *rdDepictor* para calcular las coordenadas 2D de los átomos de la molécula. Estás coordenadas son nuevas y pueden diferir significativamente de las que vienen en el archivo original. Esto es útil cuando se parte de una molécula sin coordenadas o se quiere una representación 2D estándar para la visualización o dibujo. A continuación, se muestra exactamente su uso :

```
def compute_coords(mol):
    """Ejecuta Compute2DCoords y otras sanitizaciones necesarias."""
    rdDepictor.Compute2DCoords(mol)
    return mol
```

Código 7: Recomputar coordenadas para mol en pantalla

Este paso es importante para poder generar la imagen SVG de la molécula.

4.2.2 Visualización de la molécula

La visualización de la molécula se realiza utilizando la clase *rdMolDraw2D* y específicamente el método *MolDraw2DSVG*, que genera la representación gráfica de la molécula en formato SVG a partir de sus coordenadas 2D. Esta representación es adecuada para la visualización interactiva en aplicaciones como la desarrollada en este proyecto.

El código principal que se encarga de este proceso es:

```
    def draw_molecule(self):
        """Dibuja la molécula en formato SVG y actualiza la vista."""

        problems = Chem.DetectChemistryProblems(self.mol)
        highlight_atoms  = [p.GetAtomIdx() for p in problems if hasattr(p,
'GetAtomIdx')]
        highlight_bonds  = [p.GetBondIdx() for p in problems if hasattr(p,
'GetBondIdx')]

        svg,        self.drawer        =        draw_molecule_svg(self.mol,
highlight_atoms=highlight_atoms, highlight_bonds=highlight_bonds)
        self.atom_coords_2d = get_atom_coordinates(self.drawer, self.mol)
        self.load(QtCore.QByteArray(svg.encode("utf-8")))
```

Código 8: Resalta átomos con problemas, genera y carga nuevo SVG

De forma simplificada, lo que se está haciendo es llamar a la funcón *draw_molecule_svg() (*función que contiene la lógica utilizando el método mencionado anteriormente, *MolDraw2DSVG)* que se encarga de delvolver un objeto *drawer* (utilizado para sacar las coordeandas de los elementos de la molécula) y una variable *svg* que contiene el texto SVG (que es una representación en formato XML de la imagen de la molécula).

Seguidamente, se convierte el texto SVG en una cadena de bytes codificada en UTF-8, y esta misma cadena de bytes se convierte en un objeo *QbyteArray*, que es una clase de *Qt* para manejar datos binarios.

Una vez se haya finalizado el proceso mencionado, se carga el SVG transformado para renderizar en pantalla.

4.2.3 Cálculo de coordenadas de átomos/enlaces para su manipulación

RDKit también facilita la extracción de las coordenadas de los átomos para ser utilizadas en la manipulación más adelante. En nuestro caso, el proceso es algo más exhaustivo ya que necesitamos de alguna forma poder mapear las coordenadas 2D del SVG de cada átomo a su valor "real".

Se ha mostrado anteriormente código parte de la visualización de la molécula como SVG, dentro de dicho código existe la siguiente línea:

```
. . .
self.atom_coords_2d = get_atom_coordinates(self.drawer, self.mol)
. . .
```

Código 9: Llama la función que devuelve coordenadas de átomos en SVG
La función *get_atom_coordinates* es la siguiente:

```
def get_atom_coordinates(drawer, mol):
    """Obtiene coordenadas SVG (x, y) para cada átomo de la molécula."""
    coords_svg = [drawer.GetDrawCoords(i) for i in range(mol.GetNumAtoms())]
    array_coords = np.array([[pt.x, pt.y] for pt in coords_svg])
    return array_coords
```

Código 10: Devuelve un Numpy Array con las coordenadas de los átomos

Lo único que está haciendo es calcular un array en *NumPy* [13] (que facilitará más adelante el cálculo vectorial de las diferentes coordenadas) que contiene de forma ordenada (por índices de átomos) las coordenadas 2D (x, y) de cada átomo en el SVG.

4.2.4 Identificación de átomos y enlaces cercanos

La función más relevante de la aplicación, ya que sin ella el resto de las funcionalidades no podrían operar correctamente, es la capacidad de identificar átomos o enlaces cercanos a una posición 2D específica, especialmente cuando el usuario hace clic en la imagen entre otras.

Esto se logra mediante la función *get_nearest_atom()* o *get_nearest_bond()*. Se explicará el código para *get_nearest_atom* de forma detallada para tener una idea de lo que comporta:

```
def get_nearest_atom(mol, atom_coords_2d, x_svg, y_svg):
    """Devuelve el índice del átomo más cercano y la distancia real."""
```

```
    if mol is None or mol.GetNumAtoms() == 0:
        return None, float('inf')
    target = np.array([x_svg, y_svg])
    coordinate_differences  = atom_coords_2d - target
    squared_distances = np.sum(coordinate_differences**2, axis=1)
    nearest_atom_index = np.argmin(squared_distances)
    atom_distance   =   np.sqrt(squared_distances[nearest_atom_index])      #
Distancia real (no al cuadrado)
    return nearest_atom_index, atom_distance
```

Código 10: Calcula y devulve átomo más cercano al click

La función *get_nearest_atom* tiene como propósito identificar el átomo más cercano a unas coordenadas específicas en 2D, representadas en un sistema SVG. Para ello, recibe tres parámetros principales:

- mol: la molécula que contiene los átomos,
- atom_coords_2d: un arreglo con las coordenadas 2D de todos los átomos de la molécula.
- (x_svg, y_svg): las coordenadas del punto de referencia en el sistema SVG.

Primero, la función verifica que la molécula exista y tenga átomos. Luego, crea un array *NumPy* llamado *target* con las coordenadas (x_svg, y_svg), que representa el punto en el espacio 2D con el que se compararán las posiciones atómicas.

A continuación, se calcula la diferencia entre las coordenadas de cada átomo (*atom_coords_2d*) y el punto de referencia (*target*). Esta operación es vectorizada, es decir, se realiza sobre todos los átomos de forma simultánea, generando un array de diferencias en las direcciones *X* e *Y*, lo cual ayuda a la optimización de cálculos.

Luego, se calcula la distancia euclidiana [14] al cuadrado para cada átomo. Esto se logra sumando los cuadrados de las diferencias en *X* e *Y* para cada átomo, sin calcular aún la raíz cuadrada. Trabajar con las distancias al cuadrado es más eficiente computacionalmente, ya que evita la operación costosa de la raíz cuadrada en esta etapa inicial.

La función *argmin* busca el índice del átomo con la menor distancia al cuadrado, lo que indica el átomo más cercano al punto seleccionado por el usuario.

Finalmente, para obtener la distancia real, se calcula la raíz cuadrada de la distancia al cuadrado del átomo más cercano.

La función retorna dos valores:

- El índice del átomo más cercano a las coordenadas dadas.
- La distancia real entre ese átomo y el punto de referencia.

Esta distancia se utiliza posteriormente para determinar si el objeto más cercano al punto es un átomo o un enlace, comparando sus respectivas distancias.

4.2.5 Manipulación de átomos y enlaces

Finalmente, para concluir la explicación, exploraremos como todo el proceso de manipulación de átomos y enlaces en la molécula se produce alrededor de la función *get_nearest_mol_item*, que es el punto de entrada principal para cualquier acción interactiva del usuario. Esta función se encarga de identificar qué átomo o enlace está más próximo al punto de clic, utilizando una transformación proporcional de coordenadas.

```
         def get_nearest_mol_item(self, event=None, x=None, y=None):
             """
              Devuelve el átomo o enlace más cercano al clic, si está dentro del
umbral correspondiente.
             Si no hay ninguno cerca, devuelve una coordenada SVG estimada.
             """
             atom_threshold = 6.0
             bond_threshold = 10.0

             if x is None or y is None:
                 x, y = event.x(), event.y()

             size = self.size()
             w, h = size.width(), size.height()
             viewbox = self.renderer().viewBox()
            vb_left, vb_top, vb_width, vb_height = viewbox.left(), viewbox.top(),
viewbox.width(), viewbox.height()

             x_svg = x / w * vb_width + vb_left
             y_svg = y / h * vb_height + vb_top

             # Buscar objeto más cercano
                nearest_atom_index,  atom_distance  =  get_nearest_atom(self.mol,
self.atom_coords_2d, x_svg, y_svg)
                 nearest_bond_index,  bond_distance  =  get_nearest_bond(self.mol,
self.atom_coords_2d, x_svg, y_svg)

              # Decidir cuál devolver según distancia y umbrales
             if atom_distance < bond_distance and atom_distance < atom_threshold:
                 return self.mol.GetAtomWithIdx(int(nearest_atom_index))
             elif bond_distance < bond_threshold:
                 return self.mol.GetBondWithIdx(int(nearest_bond_index))

             # Si no hay nada cerca, devolver coordenadas SVG estimadas
            return svg_to_coord(self.drawer, self.reference_points, x_svg, y_svg)
```

Código 10: Calcula y devuelve átomo más cercano al click

Antes de comenzar, hay que saber que el SVG producido por *MolDraw2DSVG* representa la molécula en un espacio lógico definido por un *viewBox*, y no en unidades de pantalla. Por lo tanto, al capturar un evento de clic del usuario, es necesario escalar las coordenadas del evento en función del tamaño actual del widget y del viewBox para determinar con precisión el punto equivalente dentro del espacio SVG.

Con las coordenadas de la molécula y del clic, se utilizan las funciones de búsqueda de la distancia más corta entre el punto de clic y los átomos/enlaces, como *get_nearest_bond* y *get_nearest_atom*, explicado anteriormente.

Si la distancia a alguno de estos elementos es menor que un umbral definido se retorna el objeto molecular correspondiente para su posterior manipulación., ya sea para eliminarlo, moverlo, cambiar su carga formal etc.

En caso de que el clic no se haya producido cerca de ningún elemento reconocible, la función devuelve una coordenada SVG estimada, que puede ser convertida al sistema de referencia interno de la molécula (2D) mediante la función *svg_to_coord* [15]. Esta conversión usa dos puntos de referencia conocidos para traducir las coordenadas SVG a coordenadas internas de la molécula, lo que permite, por ejemplo, añadir nuevos átomos en el lugar donde el usuario hizo clic.

```
     def svg_to_coord(drawer, reference_points, x_svg, y_svg):
         """Convierte coordenadas SVG a coordenadas moleculares (internas) usando
dos puntos de referencia."""
         ref0 = drawer.GetDrawCoords(reference_points[0])
         ref1 = drawer.GetDrawCoords(reference_points[1])

         dx = ref1.x - ref0.x
         dy = ref1.y - ref0.y

         if dx == 0 or dy == 0:
             raise ValueError("Los puntos de referencia no deben estar alineados
vertical u horizontalmente.")

         # Transformación inversa desde SVG a sistema de referencia molecular
         x_mol = (x_svg - ref0.x) / dx
         y_mol = (y_svg - ref0.y) / dy

         return Point2D(x_mol, y_mol)
```

Código 11: Convierte coordenadas SVG a coordenads 2D internos de la molécula

En conclusión, se quiere demostrar que se han reunido todas las funcionalidades mencionadas en un flujo coherente. La función *get_nearest_mol_item* actúa como el punto de entrada al proceso de manipulación, permitiendo identificar qué parte de la molécula (átomo o enlace) está más cerca del clic del usuario. A partir de allí, dependiendo de la acción que el usuario desee realizar (mover, agregar o eliminar un átomo o enlace), las funciones correspondientes se encargan de modificar la estructura molecular tanto en su representación gráfica como en su forma interna.

5 Implementación y algoritmos – Predictor GNN

A continuación, se profundizará en la implementación de los modelos de redes neuronales gráficas (GNN), específicamente en el modelo *GNN GINE* que ha sido el más prioritario, mientras que los 2 restantes se han utilizado como puntos de referencia para comparar el modelo previo. Se describirá el proceso de diseño y construcción del modelo, detallando cómo se define la arquitectura, qué tipo de datos se utilizan como entrada y cómo se ajustan los parámetros para optimizar su rendimiento.

Una vez explicado el modelo como tal, se pasará a la fase de entrenamiento. En esta parte, se hablará sobre los datos empleados para entrenar el modelo, la selección de las métricas de evaluación y las técnicas de optimización utilizadas. Se discutirá cómo el modelo aprende a partir de las moléculas y cómo su capacidad para predecir propiedades, como la afinidad molecular, mejora a medida que se entrena con más datos.

Finalmente, se detallará el proceso de predicción realizado por el modelo GNN una vez entrenado. En esta fase, el modelo es capaz de estimar la afinidad de la molécula en función de sus características estructurales, como la disposición de átomos y enlaces. Se explicará cómo el modelo utiliza estos parámetros para generar un valor predictivo, que refleja la relación entre la estructura de la molécula y su comportamiento esperado en aplicaciones químicas o biológicas.

5.1 Características atómicas y enlaces

Las características de los átomos y los enlaces se extraen mediante las funciones *atom_features* y *bond_features*, respectivamente. Estas funciones devuelven vectores numéricos que encapsulan propiedades clave tanto de los átomos como de los enlaces, tales como el número atómico, el grado de un átomo, la hibridación, el tipo de enlace, la conjugación, entre otras.

Cada átomo en la molécula se representa por un vector que contiene las siguientes características:

- Número atómico: Identificador único del átomo
- Grado: Número de enlaces que tiene el átomo.
- Tipo de hibridación: Especifica la disposición de los orbitales atómicos.
- Aromaticidad: Indica si el átomo forma parte de un anillo aromático.

- Número de hidrógenos: Número total de hidrógenos asociados, tanto explícitos como implícitos.
- Valencias implícitas y explícitas: Relacionadas con la capacidad del átomo para formar enlaces.
- Pertenencia a un anillo: Indica si el átomo forma parte de un anillo en la estructura molecular.
- Masa atómica: Masa del átomo en unidades de masa atómica.
- Estereoquímica: Información sobre la quiralidad del átomo (si es quiral, si tiene una configuración específica como CW o CCW).

Por otro lado, cada enlace entre dos átomos se representa por un vector con información sobre:

- Tipo de enlace: Caracteriza el enlace como simple, doble, triple, etc.
- Conjugación: Indica si el enlace forma parte de un sistema conjugado.
- Pertenencia a un anillo: Especifica si el enlace es parte de un anillo en la estructura molecular.
- Estereoquímica del enlace: Determina si el enlace es cis, trans u otra configuración estereoquímica.
- Dirección del enlace: Informa sobre la orientación direccional del enlace, lo que puede ser relevante para ciertos modelos de estereoquímica.

Estos vectores permiten representar de manera numérica las características más relevantes de los átomos y enlaces en una molécula, facilitando su uso como entrada en modelos de aprendizaje automático [16].

5.2 Normalización de datos

En este apartado, se abordará el proceso de normalización de datos en el contexto de la construcción de grafos moleculares, con un enfoque particular en la transformación y escalado de las características atómicas y de enlaces, así como de la variable objetivo en el modelo de predicción molecular.

5.2.1 Normalización de la variable objetivo (y)

Normalizar datos significa ajustar sus valores a una escala común [17]. Esto se hace para que los distintos atributos de los datos tengan un comportamiento numérico comparable. Sin normalización, una variable con valores muy grandes puede dominar el comportamiento de un modelo, aunque no sea más importante que las otras.

En el caso del modelo descrito, se tiene una variable objetivo *y* que representa una propiedad molecular (en este caso, afinidad con una proteína), la cual puede tener un rango de valores considerablemente diferente de las características de entrada (en nuestro caso este rango va de 3 a 9, aproximadamente). Si no se normaliza esta variable, el modelo podría no converger adecuadamente debido a la discrepancia en las escalas.

La función *smiles_to_data* incluye un paso de normalización de *y* utilizando un objeto scaler. Si se pasa un scaler, se utiliza para transformar la variable y al rango apropiado según las necesidades del modelo. En el código, este escalado se realiza con la siguiente línea:

```
. . .
y = scaler.transform([[y]])[0][0]
. . .
```

Código 12: Uso de Scaler para normalizar *y*

Esto indica que la variable objetivo se normaliza con el mismo scaler que se emplea para normalizar las características, de modo que todas las variables (tanto las características como la variable objetivo) tengan una distribución homogénea en cuanto a su escala.

5.2.2 Uso del StandardScaler

En el apartado 5.2.1 del código, se aplica una técnica llamada normalización de datos mediante la clase StandardScaler, que forma parte de la biblioteca sklearn.preprocessing de Python [18].

El StandardScaler transforma los datos para que cada característica tenga:

- Media (promedio) = 0
- Desviación estándar = 1

En otras palabras, toma cada valor, le resta la media de su columna y luego lo divide entre la desviación estándar de esa columna. Al realizar este paso, todos los valores están centrados alrededor de 0 (media) y tienen una dispersión ajustada (desviación estándar de 1).

5.2.3 Aplicación de la normalización en el conjunto de datos

El proceso de normalización se realiza durante la creación del conjunto de datos mediante la invocación de la función *graph_data_list*. Este método toma un *DataFrame* que contiene las representaciones SMILES y las afinidades de las moléculas, y convierte cada fila en un objeto de tipo Data, que representa un grafo molecular. La función *graph_data_lis*t se encarga de llamar a *smiles_to_data*, que, además de convertir la estructura SMILES en un grafo molecular, normaliza la variable objetivo y (afinidad) utilizando el escalador proporcionado.

```
def graph_data_list(data, scaler=None):
    graph_data_list = []
    for _, row in data.iterrows():
        d = smiles_to_data(row['smiles'], row['affinity'], scaler)
        if d:
            graph_data_list.append(d)
    return graph_data_list
```

Código 14: Extracción de carácterísticas y normalización de *y* para un DataFrame

Este enfoque garantiza que la normalización se haga de manera consistente en todo el conjunto de datos, lo que ayuda al modelo a aprender mejor. En este caso, solo se normaliza la variable objetivo y (afinidad), aunque también se podría mirar de normalizar ciertas características de los átomos. En este caso no se ha realizado, ya que la mayoría son valores discretos y ya están en un rango adecuado para usarlos en redes neuronales. Normalizar y ayuda a que el modelo converja más rápido y tenga un mejor rendimiento, especialmente porque los valores de afinidad proporcionados suelen estar entre 3 y 9, lo que es un rango bastante estrecho.

5.3 Implementación de los modelos GNN

La implementación de los modelos de redes neuronales gráficas se ha realizado utilizando *PyTorch*, junto con su extensión especializada *PyTorch Geometric*, la cual proporciona capas, estructuras de datos y funciones optimizadas para trabajar con grafos.

Para facilitar la comparación entre distintas arquitecturas, se diseñó una clase modular denominada *OptimizedGNNModel*, que permite alternar entre tres variantes ampliamente utilizadas: *GCN*, *GIN* y *GINE*. Esta estructura modular proporciona flexibilidad para la experimentación, manteniendo una arquitectura base común entre los modelos.

La arquitectura general compartida por todos los modelos se compone de varios componentes clave, que se describirán con mayor detalle en las secciones posteriores. En resumen, estos componentes incluyen:

- Las *capas convolucionales* permiten extraer características relevantes de los nodos, propagando e interpretando la información a lo largo de la estructura del grafo de acuerdo con la lógica propia de cada arquitectura (GCN, GIN o GINE).
- La normalización por lotes, *BatchNorm*, se usa para estabilizar el entrenamiento y acelerar la convergencia, lo que ayuda a controlar las distribuciones internas entre capas.
- La función de activación *ReLU*[12] se aplica tras la normalización para introducir no linealidad en el modelo, lo que permite aprender representaciones más complejas.
- Se utiliza un *pooling global* (específicamente global_mean_pool) para combinar la información de los nodos en un único vector representativo, lo cual es esencial para obtener una predicción a nivel de grafo completo.
- Para regularizar el modelo y prevenir el sobreajuste, se aplica una capa de *Dropout* con una probabilidad del X%, apagando aleatoriamente algunas neuronas durante el entrenamiento.
- Al final se utiliza una capa lineal que transforma el vector resultante en una predicción escalar, adecuada para tareas de regresión donde se estima una propiedad o valor global asociado al grafo.

[12] ReLU: Rectified Linear Unit

La salida de cada modelo es una predicción escalar por grafo, lo que lo hace apropiado para problemas como la estimación de propiedades moleculares o la predicción de métricas globales derivadas de la estructura y características del grafo. Los detalles más precisos de la implementación de cada componente serán abordados en secciones posteriores [19] [20] [21] [22].

5.3.1 Selección de arquitectura y modularidad

La clase principal del modelo recibe un argumento *model_type* que determina cuál de las tres arquitecturas utilizar. Según el valor indicado, se invoca uno de los métodos privados de construcción (*_build_gcn*, *_build_gin*, *_build_gine*), que definen las capas específicas de cada modelo:

```
. . .
if model_type == 'GCN':
    self._build_gcn(hidden_channels)
elif model_type == 'GIN':
    self._build_gin(hidden_channels)
elif model_type == 'GINE':
    self._build_gine(hidden_channels)
. . .
```

Código 16: Selección de modelo dependiendo de variable

Cada uno de estos métodos configura las capas específicas para la arquitectura seleccionada, lo que permite una comparación justa entre modelos bajo un mismo entorno experimental.

Antes de entrar en la implementación de cada uno de estos modelos, hay que saber que:

GIN es un tipo de red neuronal que trabaja con grafos y se centra en distinguir estructuras diferentes entre sí. Para hacerlo, suma las características de los nodos vecinos y las combina con las del nodo central, usando un parámetro que se aprende durante el entrenamiento. Después, esta información pasa por una red neuronal simple (MLP[13]) que ayuda a obtener representaciones más útiles. Gracias a este diseño, GIN es muy buena para tareas como clasificar grafos o predecir propiedades en moléculas.

[13] MLP: Multi-Layer Perceptron

GCN, por otro lado, es otra arquitectura de red neuronal diseñada para trabajar con grafos. A diferencia de GIN, que utiliza una agregación por suma, GCN promedia las características de los nodos vecinos. Este enfoque tiende a suavizar las representaciones, lo que puede hacer que GCN sea menos expresiva al intentar distinguir entre estructuras de grafos distintas. Sin embargo, presenta una mayor simplicidad y eficiencia computacional.

Además, GCN puede ser modificada para incorporar pesos en las aristas, lo que permite considerar, por ejemplo, el tipo o la intensidad de los enlaces. Esto facilita una ponderación diferenciada de las contribuciones de los vecinos en función de la fuerza de la conexión, aumentando su capacidad para modelar grafos con aristas informativas.

Aunque GCN es muy eficaz para tareas como la clasificación de nodos o la predicción semántica en grafos homogéneos, su capacidad discriminativa es más limitada comparada con GIN, especialmente en tareas donde se necesita distinguir con precisión entre grafos completos.

GINE es una extensión de GIN que mejora su expresividad al incorporar también información de las aristas (como se detalla en el apartado 5.1). En lugar de considerar solo las características de los nodos, GINE integra las características de las aristas en el proceso de agregación. Esto es especialmente útil en grafos donde las relaciones entre nodos contienen información relevante (por ejemplo, el tipo de enlace químico entre átomos en una molécula).

En resumen, podemos describir los modelos de la siguiente manera:

Modelo	**Agregación**	**Características Nodos**	**Características Aristas**	**Poder Expresión**	**Uso Típico**
GCN	Media (promedio normalizado)	Sí	No (solo peso)	Moderado	Clasificación de nodos y grafos simples
GIN	Suma	Sí	No	Alto	Clasificación de grafos (estructura pura)

GINE	Suma + Aritas	Sí	Sí	Más Alto que GIN	Grafos con información relevante en las aristas

Tabla 1. Comparación modelos GNN

5.3.2 Implementación del modelo GINE

Para la implementación de GINE, se entrará en más detalle que para el resto, analizando los componentes principales en la implementación del modelo, que se encuentran en el método *_build_gine* y *forward*, determinando el propósito y el motivo de su uso.

5.3.2.1 Encoder para nodos

```
. . .
self.node_encoder = nn.Linear(10, hidden_channels)
. . .
```

Código 17: Capa lineal que transforma una entrada de tamaño 10 a hidden_channels

Propósito: La capa *node_encoder* es una capa lineal que toma las características de entrada de los nodos (en este caso, 10 dimensiones por las 10 características extraídas de los nodos) y las transforma a un espacio de características ocultas (representado por *hidden_channels*). Este espacio oculto permite una representación más rica de las características de los nodos, facilitando su posterior procesamiento en la red neuronal.

¿Por qué se utiliza? El encoder para nodos es necesario para mapear las características de entrada de los nodos a un espacio de mayor dimensionalidad. Este paso es importante porque permite que las características de los nodos sean adecuadamente representadas antes de ser procesadas en la operación de convolución, donde se combinarán con las características de las aristas. La transformación a un espacio oculto más grande facilita que el modelo aprenda representaciones más complejas de los nodos y sus relaciones con los vecinos.

5.3.2.2 Encoder para aristas

```
. . .
self.edge_encoder = nn.Linear(5, hidden_channels)
. . .
```

Código 18: Capa lineal utilizada para codificar las características de las aristas.

Propósito: La capa *edge_encoder* es una capa lineal que toma las características de las aristas (en este caso, 5 dimensiones por las 5 características de las aristas) y las transforma a un espacio oculto de la misma dimensionalidad que las características de los nodos, es decir, a *hidden_channels*. Esta transformación permite que las características de las aristas sean procesadas en el mismo espacio de características ocultas que las de los nodos, facilitando su integración en el proceso de convolución.

¿Por qué se utiliza? Al incorporar las características de las aristas en el modelo, el *edge_encoder* permite al modelo aprender representaciones de las conexiones entre nodos, lo cual es crucial para tareas donde las relaciones entre los nodos son importantes (por ejemplo, en tareas de clasificación de grafos). Esta transformación asegura que las aristas sean tratadas de manera similar a los nodos en el espacio oculto.

5.3.2.3 Capas de convolución GINEConv

```
. . .
nn1 = nn.Sequential(
    nn.Linear(hidden_channels, hidden_channels),
    nn.ReLU(),
    nn.Linear(hidden_channels, hidden_channels)
)
self.conv1 = GINEConv(nn1)
. . .
```

Código 19: Secuencias de capas lineales con activación ReLU, seguida de una capa de convolución GINE

Propósito: Las capas GINEConv son el núcleo de la arquitectura GINE. La capa *nn1* es una una pequeña red neuronal (MLP) que se utiliza como función de agregación para la primera capa de convolución GINE. Esta red tiene una estructura que primero aplica una transformación lineal a las características de los nodos, seguida de una activación *ReLU*, y luego otra transformación lineal.

¿Por qué se utiliza? La elección de ReLU como función de activación es clave porque introduce no linealidades en el modelo, lo cual es necesario para aprender representaciones complejas. Si solo se usaran transformaciones lineales, el modelo sería muy limitado y no podría capturar relaciones no lineales entre los nodos y sus vecinos. En este caso se utiliza como parte de la función de agregación.

¿Por qué GINEConv? La capa GINEConv es específica de GINE y está diseñada para manejar tanto las características de los nodos como las de las aristas. Esta capa realiza la

agregación de las características de los nodos vecinos junto con las características de las aristas, lo que mejora la capacidad del modelo para diferenciar entre grafos no isomorfos.

5.3.2.4 Batch Normalization

```
. . .
self.bn1 = BatchNorm(hidden_channels)
. . .
```

Código 20: Capa de normalización por lotes para hidden_channels

Propósito: Después de cada capa de convolución, se aplica *Batch Normalization*. La normalización por lotes es una técnica para estabilizar y acelerar el entrenamiento. Ajusta la salida de cada capa para que tenga una media cercana a cero y una desviación estándar cercana a uno. Esto ayuda a prevenir problemas como el desvanecimiento o la explosión del gradiente durante el entrenamiento.

¿Por qué se utiliza?:

- Mejora la estabilidad: Al normalizar la salida de cada capa, el modelo puede entrenarse más rápido y con mayor estabilidad.
- Evita el sobreajuste: Al mejorar la generalización, se reduce la posibilidad de que el modelo se ajuste demasiado a los datos de entrenamiento.
- Facilita el entrenamiento de redes: Aunque en este caso la red tiene solo dos capas, *Batch Normalization* también ayuda. Sirve para que los valores que se usan para aprender (los gradientes) no se vuelvan ni muy pequeños ni muy grandes. Esto hace que el modelo aprenda de forma más estable y rápida.

5.3.2.5 Función de activación ReLU

```
...
x = torch.relu(x)
...
```

Código 21: Aplicación de la función de activación ReLU sobre *x*

Propósito: Después de cada operación de convolución y normalización, se aplica la función de activación ReLU. Esta función es una de las más utilizadas en redes neuronales

debido a su simplicidad y efectividad. ReLU transforma todos los valores negativos en cero y deja pasar sin cambios los valores positivos.

¿Por qué se utiliza?:

- Introduce no linealidades: ReLU es una función no lineal, lo que permite que la red aprenda patrones y representaciones complejas en los datos. Sin funciones no lineales, las redes profundas no tendrían mayor capacidad que una simple combinación lineal. En este caso se aplica después de cada *Batch Normalization*, sobre los *embeddings* actualizados del nodo.
- Evita el problema de los gradientes desvanecidos: En algunas funciones más antiguas como sigmoid o tanh, cuando los valores de entrada son muy grandes o muy pequeños, las salidas cambian muy poco. Eso hace que el modelo aprenda muy lento, porque los cambios que necesita hacer (los gradientes) se vuelven muy pequeños o casi cero.

  ReLU, en cambio, mantiene valores grandes de gradiente para las entradas positivas, lo que permite que el modelo aprenda más rápido y mejor, especialmente cuando tiene muchas capas.
- Simplicidad y eficiencia computacional: ReLU es muy eficiente ya que solo requiere comparar con cero. sto la hace ideal para redes neuronales modernas que requieren muchas operaciones.

5.3.2.6 Global Mean Pooling

```
...
x = global_mean_pool(x, batch)
...
```

Código 22: Aplicación de pooling global por promedio sobre las representaciones x utilizando los índices de batch

Propósito: El global mean pooling toma la representación de todos los nodos de un grafo y las combina en un solo vector representativo. En lugar de utilizar otros metodos, como la suma o el máximo, se realiza un promedio global de las características de todos los nodos.

¿Por qué se utiliza?:

- Condensación de información: Al promediar todas las características de los nodos, el modelo obtiene una representación compacta del grafo.

- Desempeño en clasificación de grafos: Esta operación es útil para tareas de clasificación de grafos, donde el objetivo es representar un grafo completo (en lugar de un nodo específico).
- Simplicidad y efectividad: Global pooling permite mantener la complejidad computacional controlada, especialmente en grafos grandes, y funciona bien en la mayoría de las tareas de GNN.

5.3.2.6.1 Flujo completo de Forward

El *forward* en un GNN es el proceso por el cual los datos de entrada, como las características de los nodos y aristas, se "pasan hacia adelante" a través de las capas de la red para producir una salida, como las representaciones de los nodos. Durante este paso, cada capa del GNN realiza operaciones de agregación y transformación de la información de los nodos y sus vecinos, generando nuevas representaciones que capturan la estructura y relaciones del grafo.

Cuando se dice que los datos de entrada se "pasan hacia adelante", nos referimos a como la información inicial se va procesando capa por capa definida para ir generando nuevas representaciones o predicciones.

```
...
x = self.node_encoder(x)
edge_attr = self.edge_encoder(edge_attr)

x = self.conv1(x, edge_index, edge_attr)
x = self.bn1(x)
x = torch.relu(x)

x = self.conv2(x, edge_index, edge_attr)
x = self.bn2(x)
x = torch.relu(x)
...
```

Código 22: Propagación de nodos y aristas con convolución, normalización y activación.

1. Transformación de nodos y aristas:
    - Primero, las características de los nodos son transformadas usando node_encoder, y las características de las aristas son transformadas con edge_encoder.
2. Convolución GINEConv:

- Las características de los nodos y las aristas pasan a través de dos capas de convolución GINEConv, donde cada capa es seguida por una normalización por lotes (BatchNorm) y activación ReLU.

3. Global Mean Pooling y Capa Final:
    - Después de procesar las características a través de las convoluciones, se realiza el *global mean pooling* para obtener una representación global del grafo, que luego pasa a través de una capa final Linear para hacer la predicción.

5.3.3 Implementación del modelo GIN

El modelo GIN está diseñado para mejorar la capacidad de los modelos de redes neuronales gráficas en la discriminación de grafos no isomorfos, es decir mejora la capacidad del GNN para distinguir grafos que no son isomorfos, es decir, que tienen estructuras diferentes, lo que es clave para tareas donde se deben identificar grafos similares pero no idénticos.

En esta implementación, la principal característica de GIN es el uso de una red neuronal para la agregación de las características de los nodos. En lugar de utilizar una simple suma o promedio de las características de los vecinos, como en modelos como GCN, GIN emplea una pequeña red neuronal (MLP) que permite una mayor flexibilidad en la forma en que se combinan las características de los nodos y sus vecinos. Esta estructura le permite aprender representaciones más complejas y poderosas.

Al igual que otros modelos de redes neuronales, se aplican técnicas como normalización por lotes y activación ReLU para garantizar un entrenamiento estable y mejorar el rendimiento general. Tras la aplicación de las capas de convolución, las representaciones de los nodos se agregan globalmente usando una operación de global mean pooling, lo que produce una representación única para el grafo completo, que luego puede ser utilizada en tareas de clasificación o regresión.

5.3.4 Implementación del modelo GCN

El modelo GCN es uno de los enfoques más populares y sencillos en redes neuronales gráficas, basado en la agregación de características de los nodos vecinos de manera directa.

En esta implementación, el modelo utiliza capas de convolución GCNConv que permiten a cada nodo aprender una representación combinando sus características y las de sus vecinos usando una media ponderada.

El proceso es relativamente directo: cada nodo toma las características de los nodos vecinos, las combina y aplica una transformación que depende de las conexiones del grafo, lo que le permite aprender representaciones más informativas de la estructura local del grafo. Al igual que en GIN, se emplean técnicas de Batch Normalization y ReLU para asegurar la estabilidad durante el entrenamiento.

Después de las capas de convolución, las características de todos los nodos son agregadas globalmente mediante una operación de global mean pooling, que genera una representación del grafo en su totalidad. Esta representación es luego utilizada para realizar una predicción sobre el grafo o para tareas de clasificación.

5.4 Entrenamiento de los modelos GNN

En este apartado, se describe el proceso de entrenamiento de los modelos anteriormente mencionados, utilizados en este trabajo. Este proceso se implementó utilizando principalmente, una vez más, las bibliotecas *PyTorch* y *PyTorch Geometric*, herramientas ampliamente utilizadas para trabajar con redes neuronales gráficas.

5.4.1 Conceptos claves en el entrenamiento

Antes de entrar en los detalles específicos del entrenamiento, es importante entender algunos conceptos clave relacionados con el proceso de optimización y aprendizaje:

AdamW [23]: El optimizador AdamW es una variante del optimizador Adam, con la diferencia principal de que aplica un decaimiento de peso (weight decay) explícito. Este decaimiento ayuda a regularizar el modelo, reduciendo la posibilidad de sobreajuste. La fórmula de AdamW ajusta las tasas de aprendizaje basadas en estimaciones de primer y segundo momento de los gradientes. En esta implementación, se utiliza una tasa de aprendizaje de *5e-4* y un peso de decaimiento de *5e-5*.

- *Tasa de Aprendizaje (lr)*: La tasa de aprendizaje es un parámetro crucial que controla qué tan grandes son los ajustes que se hacen a los pesos durante cada paso de optimización. Una tasa demasiado alta puede hacer que el modelo

diverja, mientras que una tasa demasiado baja puede resultar en un entrenamiento excesivamente lento.

- *Decaimiento de Peso (Weight Decay)*: Es una técnica de regularización que penaliza los valores grandes en los parámetros del modelo, evitando que el modelo se sobreajuste a los datos de entrenamiento. Esto se logra añadiendo un término adicional al valor de la función de pérdida que penaliza el valor de los pesos del modelo.

ReduceLROnPlateau [24]: Este scheduler ajusta la tasa de aprendizaje durante el entrenamiento en función de la pérdida de validación. En particular, cuando la pérdida no mejora después de un número determinado de épocas, la tasa de aprendizaje se reduce automáticamente. Esto ayuda a que el modelo refine sus parámetros de manera más eficiente cuando ya no está haciendo mejoras significativas con la tasa de aprendizaje actual. En este caso, se utiliza con un factor=0.5 y un patience=10, lo que significa que la tasa de aprendizaje se reduce a la mitad si la pérdida no mejora durante 10 épocas consecutivas.

Early Stopping [24]: Es una técnica utilizada para evitar el sobreentrenamiento del modelo. Si la pérdida de validación no mejora durante un número predefinido de épocas consecutivas (patience), el proceso de entrenamiento se detiene para evitar un ajuste excesivo a los datos de entrenamiento. En este trabajo, el umbral de patience se establece en 20 épocas.

Función de Pérdida [25] *(Loss Function)*: Para este tipo de tareas de regresión, se utiliza el error cuadrático medio, que calcula la media de los errores cuadrados entre las predicciones del modelo y las etiquetas verdaderas.

5.4.2 Descripción del Proceso de Entrenamiento

Como se mencionó en la introducción, para entrenar los modelos se utilizó la base de datos URV-May-2024, que recopila inhibidores no covalentes de la proteasa principal del SARS-CoV-2 junto con sus valores de IC50. Esta base contiene 233 compuestos, cada uno

con su estructura molecular y un valor que indica su capacidad para inhibir la enzima. También se calculó el valor de pIC50, que permite comparar la potencia entre compuestos. En este trabajo, solo se utilizaron las estructuras de los inhibidores y sus valores de actividad, sin incluir información de la proteína.

El proceso de entrenamiento del modelo GNN comienza con la carga y preparación de los datos. Las muestras moleculares, representadas por cadenas SMILES, se procesan inicialmente para filtrar las variables relevantes, como la afinidad, que es la propiedad que queremos predecir. Para garantizar que las etiquetas de afinidad sean adecuadas para el modelo, se normalizan utilizando un *StandardScaler*. Esta normalización es crucial, ya que facilita la convergencia durante el entrenamiento y mejora la estabilidad del proceso de aprendizaje. El escalador resultante se guarda, lo que permitirá reutilizarlo en el futuro para procesar los datos de manera consistente durante la inferencia.

A continuación, cada molécula se convierte en una representación gráfica, lo cual es esencial para que los modelos GNNs puedan aprender las relaciones estructurales dentro de las moléculas. Estas representaciones gráficas se generan utilizando la función *graph_data_list*, apartado 5.2.3, que convierte las cadenas SMILES en grafos que el modelo puede procesar.

Una vez que los datos están listos, se dividen en tres partes: un 70% se usa para entrenar el modelo, un 15% para validar su desempeño durante el entrenamiento y otro 15% para probar qué tan bien generaliza al final. Esta división permite controlar el aprendizaje del modelo sin evaluarlo sobre los mismos datos con los que fue entrenado. Luego, todos estos conjuntos se cargan en un DataLoader de PyTorch Geometric. De forma sencilla, este componente se encarga de organizar los datos en mini-lotes (mini-batches), que se procesan en cada paso del entrenamiento. Esto no solo hace que el modelo entrene más rápido al aprovechar mejor la memoria (especialmente en GPU), sino que también ayuda a evitar sesgos, ya que los datos se barajan en cada época. Además, los mini-batches permiten una mejor paralelización y aportan mayor estabilidad durante el aprendizaje, ya que el modelo ajusta sus parámetros en base a múltiples ejemplos a la vez en lugar de uno solo.

En cuanto al modelo, se utiliza una instancia de la clase *OptimizedGNNModel*, apartado 5.3 y 5.3.1, cuya arquitectura se elige antes de comenzar el entrenamiento. El modelo se entrena en el dispositivo disponible, ya sea GPU o CPU, dependiendo de la infraestructura de hardware. Durante el proceso de optimización, se emplea el optimizador *AdamW*. Este decaimiento ayuda a regularizar el modelo, evitando el sobreajuste al penalizar los valores grandes en los parámetros del modelo. El optimizador ajusta automáticamente las tasas de aprendizaje, lo que mejora la eficiencia en el proceso de entrenamiento.

Además, para gestionar la tasa de aprendizaje durante el entrenamiento, también se utiliza un scheduler llamado *ReduceLROnPlateau*. Este ajusta la tasa de aprendizaje durante el entrenamiento en función de la evolución de la pérdida. Si la pérdida no mejora después de un número determinado de épocas, la tasa de aprendizaje se reduce automáticamente, lo que permite al modelo afinar sus parámetros de manera más precisa en las etapas finales del entrenamiento.

El criterio utilizado para evaluar el desempeño del modelo es el error cuadrático medio, que mide la diferencia entre las predicciones del modelo y las etiquetas reales. Este criterio es adecuado para la tarea de regresión que estamos abordando, ya que estamos prediciendo una propiedad numérica continua (afinidad) a partir de las estructuras moleculares.

El modelo se entrena por un máximo de 300 épocas. No obstante, para evitar el sobreentrenamiento y reducir el uso innecesario de recursos computacionales, se emplea una técnica de Early Stopping. Si la pérdida de validación no mejora durante 20 épocas consecutivas, el entrenamiento se detiene de forma anticipada. Cada vez que el modelo alcanza un nuevo mínimo en la pérdida de validación, se guarda su estado como la mejor versión hasta ese momento.

Durante el proceso, se registran las pérdidas promedio por época tanto en entrenamiento como en validación. Estos datos se visualizan en un gráfico que permite evaluar si el modelo está convergiendo adecuadamente, identificar posibles problemas de sobreajuste o detectar la necesidad de ajustar la tasa de aprendizaje.

Una vez finalizado el entrenamiento, se carga el modelo con mejor rendimiento y se evalúa su desempeño en el conjunto de prueba. Esta evaluación final proporciona una estimación objetiva de la capacidad de generalización del modelo, con base en muestras no vistas durante el entrenamiento ni la validación.

El siguiente fragmento de código muestra el núcleo del proceso de entrenamiento, donde se procesan los datos, se calcula la pérdida y se actualizan los parámetros del modelo:

A continuación, se muestra un fragmento representativo del bucle de entrenamiento:

```
. . .
for epoch in range(1, max_epochs + 1):
    model.train()
    total_train_loss = 0

    for batch in train_loader:
        batch = batch.to(device)
        optimizer.zero_grad()
        pred = model(batch)

        loss = loss_fn(pred, batch.y.view(-1))
        loss.backward()
        optimizer.step()

        total_train_loss += loss.item() * batch.num_graphs

    avg_train_loss = total_train_loss / len(train_loader.dataset)
    train_losses.append(avg_train_loss)

    # Validación
    model.eval()
    total_val_loss = 0
    with torch.no_grad():
        for batch in val_loader:
            batch = batch.to(device)
            pred = model(batch)
            loss = loss_fn(pred, batch.y.view(-1))
            total_val_loss += loss.item() * batch.num_graphs

    avg_val_loss = total_val_loss / len(val_loader.dataset)
    val_losses.append(avg_val_loss)

    scheduler.step(avg_val_loss)

    print(f"Epoch {epoch}, Train Loss: {avg_train_loss:.4f}, Val
Loss: {avg_val_loss:.4f}")

    epochs_trained += 1

    # Early stopping basado en validación
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        patience_counter = 0
```

```
            torch.save(model.state_dict(),       model_save_path       +
f'{model_type}_{model_number}.pth')
            print(f"New best model saved at epoch {epoch}")
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print(f"Early stopping triggered at epoch {epoch}")
                break
    . . .
```

Código 24: Entrenamiento del modelo con cálculo de pérdida, optimización y control de paciencia.

6 Evaluación y resultados

La evaluación de este proyecto se llevará a cabo de manera diferencia: la interfaz gráfica de la aplicación y el rendimiento de los modelos GNN.

Por un lado, se evaluará si la interfaz gráfica cumple con los requisitos funcionales y técnicos establecidos en el diseño, asegurando que los usuarios puedan interactuar con el sistema de manera eficiente y efectiva.

Por otro lado, se realizará una evaluación específica para los modelos GNN, centrada en su capacidad para predecir con precisión la afinidad entre los ligandos y las proteínas. En esta parte, se evaluará la exactitud de las predicciones realizadas por el modelo, así como su rendimiento en términos de generalización y robustez ante datos de entrada variados.

6.1 Evaluación de la interfaz gráfica

Prueba	**Resultado esperado**	**Resultado obtenido**
Cargar molécula desde fichero SDF	El sistema carga la molécula correctamente y la muestra en la interfaz.	Correcto
Crear nueva molécula a partir de canvas vacío	El sistema crea una nueva molécula vacía o con un átomo nuevo.	Correcto
Añadir átomo a partir de botón "Añadir átomo/enlace" en posición correcta	El sistema crea un nuevo átomo a partir del click del usuario.	Correcto
Añadir átomo a partir de botón "Añadir átomo/enlace" en posición incorrecta	El sistema no crea el nuevo átomo por la posición incorrecta del usuario, demasiado cerca de otro átomo/enlace	Correcto
Añadir átomo a partir de seleccionar un elemento en el menú	El sistema crea un nuevo átomo a partir del click del usuario.	Correcto
Añadir átomo a partir de seleccionar un elemento en el menú en posición incorrecta	El sistema no crea el nuevo átomo por la posición incorrecta del usuario, demasiado cerca de otro átomo/enlace	Correcto
Añadir enlace entre dos átomos sin enlace	El sistema crea un enlace entre átomos y actualiza la	Correcto

	visualización correctamente.	
Añadir enlace entre dos átomos con enlace	El sistema no crea el enlace entre los átomos.	Correcto
Eliminar átomo o enlace	El sistema elimina el átomo o enlace seleccionado y actualiza la visualización correctamente.	Correcto
Cambiar tipo de átomo o enlace	El sistema cambia el tipo de átomo o enlace correctamente y actualiza la visualización.	Correcto
Cambiar carga formal del átomo	El sistema cambia la carga formal del átomo seleccionado y actualiza la visualización.	Correcto
Mover átomo	El sistema mueve el átomo seleccionado a la nueva posición y actualiza la visualización correctamente.	Correcto
Mostrar propiedades del átomo	El sistema muestra las propiedades detalladas del átomo seleccionado.	Correcto
Recalcular coordenadas correctamente	El sistema recalcula y actualiza las coordenadas de la molécula correctamente.	Correcto
Recalcular coordenadas cuando no existe ninguna molécula	El sistema recalcula y actualiza las coordenadas de la molécula correctamente.	Correcto
Sanitizar la molécula correctamente	El sistema sanitiza la molécula correctamente.	Correcto
Sanitizar la molécula cuando es invalida	El sistema intenta sanitizar la molécula y falla en el proceso.	Correcto
Añadir hidrógenos explícitos	El sistema añade hidrógenos explícitos a la molécula y actualiza la visualización correctamente.	Correcto

Añadir hidrógenos explícitos en molécula con hidrógenos visibles	El sistema añade hidrógenos explícitos a la molécula y actualiza la visualización correctamente.	Correcto
Quitar hidrógenos explícitos	El sistema elimina los hidrógenos explícitos de la molécula y actualiza la visualización correctamente.	Correcto
Quitar hidrógenos explícitos en molécula con error	El sistema elimina los hidrógenos explícitos de la molécula teniendo un error.	Incorrecto
Limpiar Canvas con molécula	El sistema limpia el canvas y elimina cualquier molécula mostrada.	Correcto
Guardar molécula como fichero SDF	El sistema guarda la molécula correctamente como un fichero SDF.	Correcto
Guardar molécula con errores como fichero SDF	El sistema da un aviso para asegurarse de que el usuario quiera guarar el archivo con errores.	Correcto
Guardar molécula vacia como fichero SDF	El sistema da un aviso por la barra de status sobre la falta de contenido	Correcto
Exportar imagen PNG	El sistema guarda la molécula correctamente como un fichero PNG.	Correcto
Predecir afinidad con modelos GNN	El sistema calcula y muestra la predicción de afinidad correctamente basada en el modelo GNN.	Correcto

Tabla 2. Tabla de juego de pruebas

6.2 Evaluación de los modelos GNN

Para evaluar el desempeño de los modelos, se utiliza como métrica principal la función de pérdida (loss), que cuantifica el error entre las predicciones del modelo y los valores reales. Una pérdida baja indica que el modelo está ajustando bien los datos, mientras que una pérdida alta señala que el modelo comete errores significativos. Sin embargo, es importante interpretar estas pérdidas en los distintos conjuntos (entrenamiento, validación y test) para evaluar correctamente la capacidad de generalización del modelo.

Antes de entrar en el detalle de la evaluación de los modelos, es importante tener en cuenta, como se mencionó en el apartado 5.4.2, que el entrenamiento se realizó utilizando un 70% para entrenamiento, 15% para validación y 15% para prueba.

Con esto en mente, es crucial tener en cuenta lo siguiente:

Dado que el dataset proporcionado es bastante pequeño (aproximadamente ~240 muestras) el hecho de reservar una parte de él para el set validación y/ prueba reduce considerablemente la cantidad de datos disponibles para entrenar, lo cual puede afectar negativamente la capacidad del modelo.

Se han empleado diferentes estrategias para mitigar el sobreajuste, es decir, evitar que el modelo aprenda demasiado bien los datos de entrenamiento y no logre generalizar, en particular aquellas mencionadas previamente:

- El *early stopping* detiene el entrenamiento cuando la pérdida en el conjunto de validación deja de mejorar tras un número determinado de épocas. Aunque está métrica no mide directamente la capacidad de generalización, puede funcionar bien en combinación con otras técnicas.
- Las técnicas de regularización ayudan a mitigar el sobreajuste, incorporando diferentes métodos como el *dropout*, *weight decay* y la reducción adaptativa de la tasa de aprendizaje (*lr*), limitan la complejidad del modelo y controla la convergencia del entrenamiento.

Para comenzar con la evaluación de los modelos, tenemos los siguientes gráficos generados después del entrenamiento de cada modelo representando su perdida a través de los ciclos o epoch:

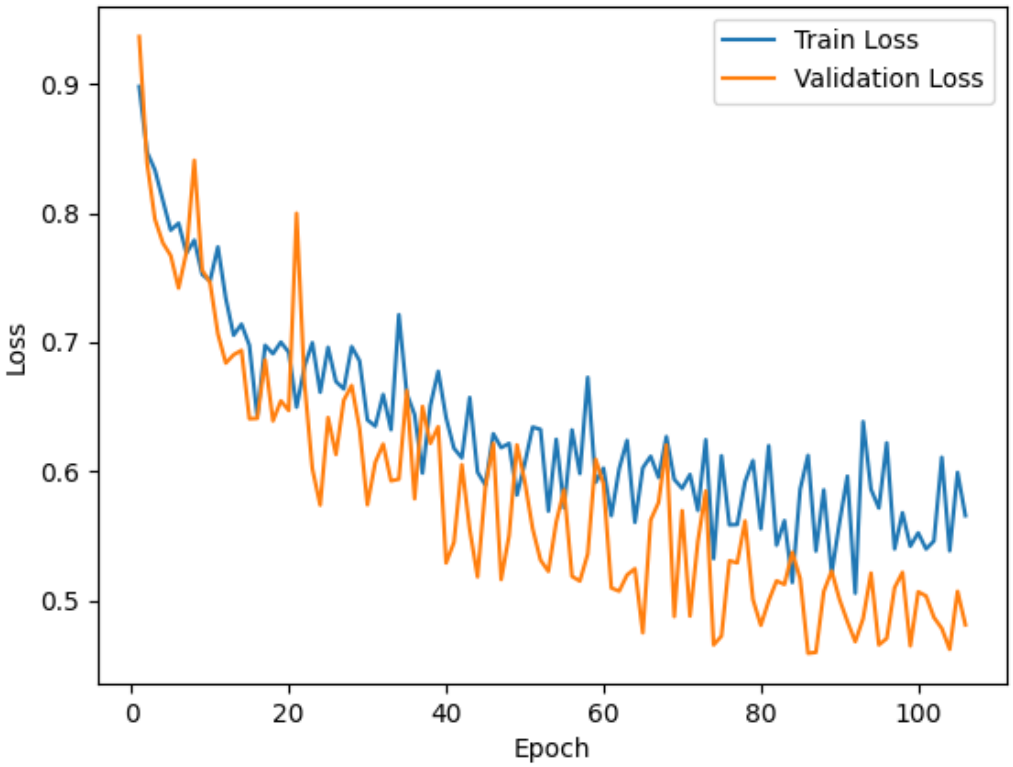

Figura 13. Representación gráfica de la perdida para GCN

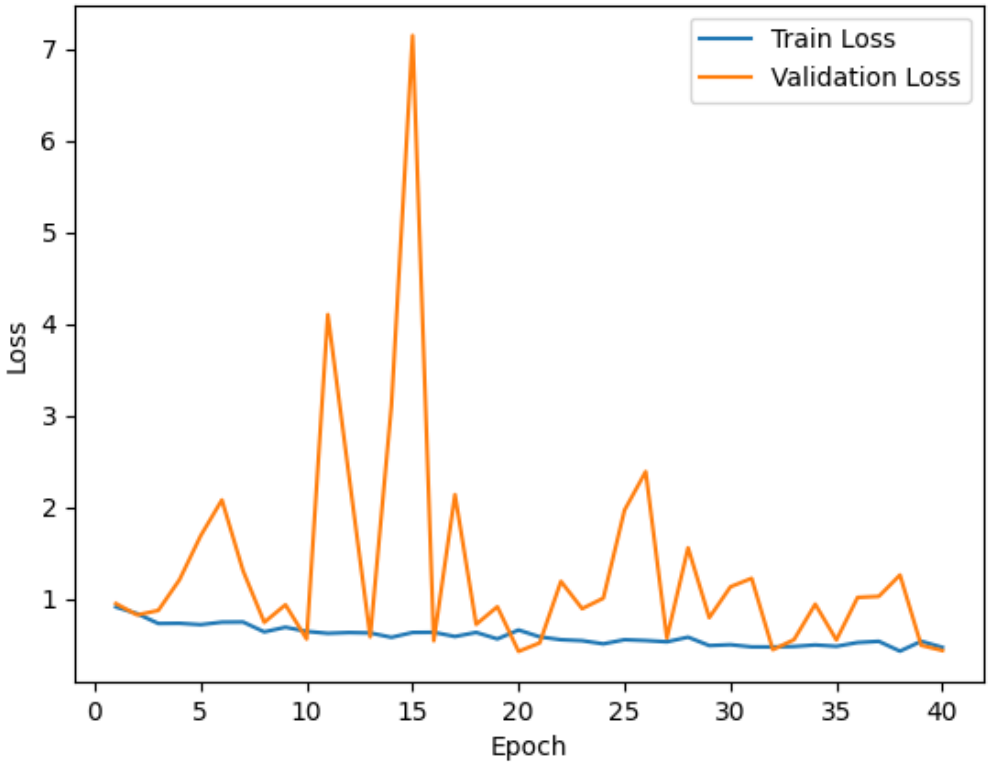

Figura 14. Representación gráfica de la perdida para GIN

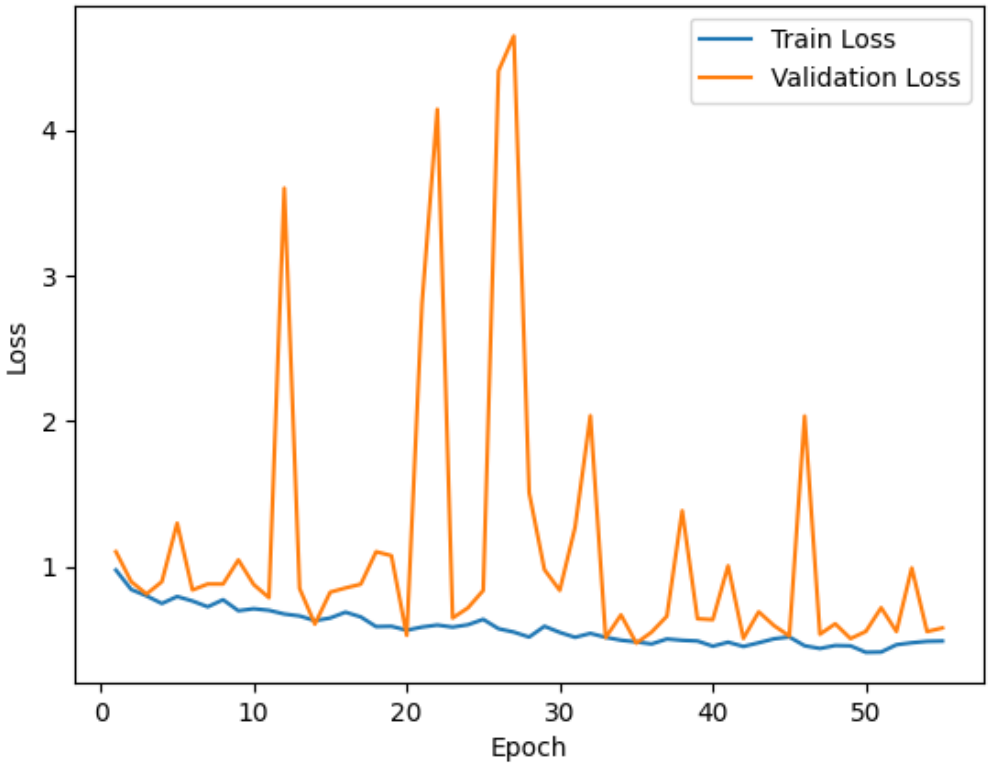

Figura 15. Representación gráfica de la perdida para GINE

El siguiente análisis compara las curvas de pérdida durante el entrenamiento de los tres modelos GNN, con el objetivo de evaluar su rendimiento y eficiencia en la reducción de la función de pérdida. Se analiza cómo evoluciona la pérdida al inicio y al final del entrenamiento, su estabilidad y la tendencia general de mejora, utilizando gráficas como apoyo visual.

Además, se realiza una comparación para identificar cuál de los modelos converge de forma más eficiente. Finalmente, se discute cuál modelo resulta más adecuado según su comportamiento durante el entrenamiento.

Para el modelo *GCN (*Figura 13*)*:

- Tendencia: Descendente suave y estable tanto para la pérdida de entrenamiento como de validación, con ligeras oscilaciones. La validación mejora consistentemente.

- Pérdida entrenamiento final (~época 106): ~0.5657
- Pérdida validación final (~época 106): ~0.4812
- Pérdida test final: ~0.9544
- Comportamiento: Comportamiento estable y predecible. No hay signos claros de sobreajuste ni inestabilidad. El modelo converge lentamente, pero de manera segura. Poca capacidad de generalización basado en la pérdida del test.

Para el modelo GIN (Figura 14):

- Tendencia: Volátil en validación, con picos frecuentes. La pérdida de entrenamiento disminuye de forma estable, pero la validación muestra alta variabilidad.
- Pérdida final (~época 40): ~0.4705
- Pérdida validación final (~época 40): ~0.4361
- Pérdida test final: ~0.9589
- Comportamiento: Inestable en validación. Aunque alcanza bajas pérdidas, las oscilaciones en validación sugieren que el modelo es sensible a pequeñas variaciones en los datos, posiblemente sobreajustando en ciertos puntos.

Para el modelo GINE (Figura 15):

- Tendencia: Similar a GIN, con validación errática. La pérdida de entrenamiento es baja y estable, pero la validación muestra picos elevados y frecuentes.
- Pérdida final (~época 55): ~0.4921
- Pérdida validación final (~época 55): ~0.5815
- Pérdida test final: ~0.8761
- Comportamiento: Volatilidad marcada en la validación. Aunque logra una buena generalización en test, el entrenamiento muestra señales claras de inestabilidad durante la validación, lo que sugiere sensibilidad al conjunto de validación.

De forma más estructurada, tenemos:

Modelo	**Pérdida entrenamiento final**	**Pérdida validación final**	**Pérdida test final**	**Nº épocas**	**Comportamiento general**
GCN	0.5657	0.4812	0.9544	~106	Estable, menor capacidad de generalización
GIN	0.4705	0.4361	0.9589	~40	Volatibilidad con buena validación, mala capacidad de generalización
GINE	0.4921	0.5815	0.8761	~55	Volatibilidad en validación, buena generalización

Tabla 3. Comparación estructurada de los modelos

Aunque GIN obtiene las pérdidas más bajas tanto en entrenamiento como en validación, su rendimiento en el conjunto de test no refleja una buena capacidad de generalización. Esto sugiere que el modelo podría estar sobreajustando los datos de entrenamiento, mostrando un buen ajuste interno pero perdiendo eficacia con datos nuevos.

GCN, por su parte, necesita muchas más épocas para converger y, aunque su rendimiento en validación no es el mejor, consigue una pérdida en test similar a GIN. No parece destacar especialmente en ningún aspecto, pero su comportamiento es más predecible y progresivo durante el entrenamiento.

El modelo GINE no logra las mejores métricas en entrenamiento ni en validación, pero consigue la mejor pérdida en el conjunto de test, lo que indica una mayor capacidad de generalización. Esto lo convierte en el modelo más sólido para predecir sobre datos no vistos, a pesar de que su rendimiento durante el entrenamiento pueda parecer menos llamativo.

En resumen, aunque GIN presenta el mejor ajuste durante las fases iniciales, GINE demuestra ser el modelo más fiable en términos de rendimiento final sobre datos desconocidos. GCN, si bien es estable, no logra destacar en precisión. Por tanto, GINE sería la opción más adecuada si se prioriza la capacidad de generalización por encima del ajuste estricto durante el entrenamiento.

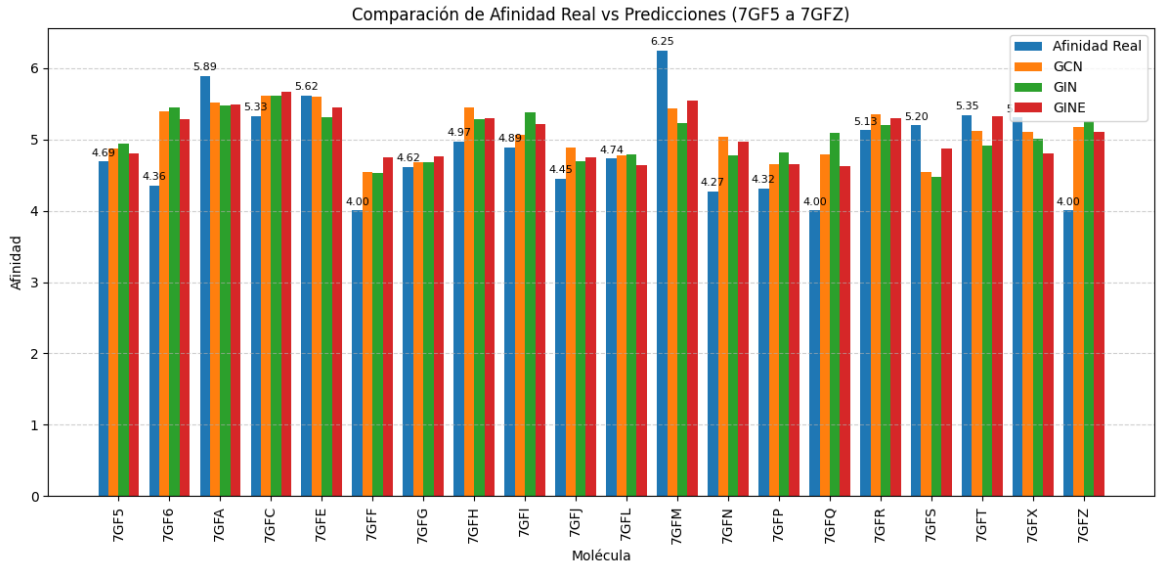

Figura 16. Comparación de los valores predecidos

En la Figura 16 se muestra una comparación visual entre la afinidad real de varias moléculas y las predicciones generadas por tres modelos distintos de redes neuronales de grafos: GCN, GIN y GINE. Cada línea representa la evolución de la afinidad a lo largo de una serie de moléculas identificadas por sus códigos (por ejemplo, 7GF5, 7GFA, etc.). La afinidad real está representada por barras azules, mientras que las predicciones de los modelos se muestran en naranja (GCN), verde (GIN) y rojo (GINE).

En general, dentro de lo que cabe, todos los modelos siguen de forma razonable la tendencia de la afinidad real, aunque con diferentes grados de precisión y en su mayoría nunca tan cerca del valor real. En varios casos, como las moléculas 7GF5, 7GFG o 7GFR, las predicciones están bastante próximas al valor real. Sin embargo, en otras como 7GFA,

7GFM o 7GFZ, se observan discrepancias más marcadas, lo que sugiere que ciertas estructuras moleculares son más difíciles de modelar para estas arquitecturas.

Es importante destacar que en la Figura 16 se muestra únicamente una selección de 20 moléculas, de un total aproximado de 240. Por tanto, este análisis visual proporciona solo una visión muy reducida del comportamiento de los modelos. Aunque en esta muestra puede parecer que ciertos modelos se ajustan mejor en determinados ligandos, esta percepción puede no coincidir con el rendimiento global.

De hecho, los resultados cuantitativos obtenidos a partir de todo el conjunto de datos, mediante las métricas MAE y RMSE, indican que el modelo GINE es el que ofrece mayor precisión general. En concreto, GINE alcanza un MAE de 0.6091 y un RMSE de 0.7721, superando tanto a GCN (MAE: 0.6369, RMSE: 0.8064) como a GIN (MAE: 0.6551, RMSE: 0.8423). Esto sugiere que, a pesar de las posibles discrepancias visuales en un subconjunto limitado, GINE demuestra ser el modelo más robusto cuando se consideran todas las predicciones en su conjunto.

Métrica	**GCN**	**GIN**	**GINE**
MAE	0.6369	0.6551	0.6091
RMSE	0.8064	0.8423	0.7721

Tabla 4. Cálculo MAE y RMSE para las GNN (más bajo mejor)

En conjunto, el gráfico permite observar que, si bien los modelos de GNN utilizados logran capturar ciertos patrones en el comportamiento de la afinidad molecular frente a la proteasa del SARS-CoV-2, todavía presentan diferencias significativas en su capacidad para predecir con precisión los valores de entrenamiento, y más aún en lo que respecta a su capacidad de generalización. Estas diferencias podrían atribuirse a diversos factores; en nuestro caso, principalmente al tamaño reducido del conjunto de datos disponible para el entrenamiento. No obstante, también deben considerarse otros aspectos relevantes, como la

arquitectura de los modelos, la representación de las moléculas, la calidad de los datos de entrada o el preprocesamiento aplicado.

7 Legislación y protección de datos

En este proyecto no se trata con información personal ni datos sensibles de ningún tipo. La aplicación está diseñada para funcionar de forma local en el equipo del usuario, sin almacenar datos externos ni enviar información a servidores externos o bases de datos. Los datos utilizados, que son estructuras moleculares y sus propiedades químicas, son proporcionados directamente por el usuario durante la ejecución y se eliminan automáticamente una vez finalizada la sesión.

Por tanto, no existe riesgo relacionado con la privacidad o protección de datos personales en el marco de este desarrollo. No se recogen, procesan ni almacenan datos personales, ni se emplean técnicas de análisis que impliquen información identificable de individuos. Esto simplifica el cumplimiento de normativas de protección de datos como el RGPD (Reglamento General de Protección de Datos), ya que el proyecto no maneja datos que estén sujetos a estas regulaciones.

En resumen, el proyecto garantiza la privacidad y confidencialidad de los usuarios al operar exclusivamente con datos científicos anónimos y temporales, respetando las buenas prácticas en materia de protección de datos y privacidad informática.

8 Implicaciones éticas, de igualdad y medioambientales

8.1 Igualdad

En cuanto a la igualdad, el proyecto no maneja datos personales ni información demográfica, por lo que no existe riesgo de sesgos de género o discriminación en los datos utilizados. Dado que el proyecto trabaja con datos moleculares anónimos y propiedades químicas, este riesgo es mínimo. No obstante, la interfaz gráfica ha sido diseñada para ser accesible y usable para cualquier persona, sin distinción de género ni otras características, garantizando igualdad en la experiencia de usuario.

8.2 Medio ambiente

El desarrollo del proyecto se centra en una aplicación de software, que por su naturaleza tiene un impacto ambiental indirecto, principalmente asociado al consumo energético durante el uso del hardware para entrenamiento y ejecución. Para minimizar este impacto, se han implementado técnicas de optimización en el entrenamiento de los modelos para evitar un uso excesivo de recursos computacionales y reducir el tiempo de cómputo.

8.3 Responsabilidad social.

El proyecto aporta un valor social al facilitar el desarrollo de herramientas para la investigación biomédica, concretamente en la manipulación y predicción de afinidad molecular contra la proteasa prioncipal del SARS-CoV-2. La aplicación es de libre uso local, promoviendo la transparencia y accesibilidad sin necesidad de recursos costosos o infraestructuras complejas.

8.4 Ética

Desde el punto de vista ético, el proyecto respeta la privacidad de los usuarios al no recolectar ni almacenar datos personales. Asimismo, se ha desarrollado bajo principios de transparencia y ha mantenido una actitud crítica en la interpretación de los resultados.

9 Conclusiones

El objetivo principal de desarrollar y crear una interfaz gráfica funcional que permita la edición de moléculas interactivamente, bajo mi opinión, se ha cumplido satisfactoriamente.

Dentro de lo que se propuso inicialmente, la capacidad de poder añadir/eliminar átomos y enlaces ha sido llevado a cabo de forma correcta, e incluso añadiendo funcionalidades que no tenía completadas inicialmente, como simple hecho de poder ver diferentes características de los átomos al pasar el ratón por encima, o poder mover dichos átomos arrastrándolo sobre el canvas, que le dan un valor añadido a la aplicación, mejorando la sensación de interactividad y usabilidad.

Sin embargo, debo reconocer que mi conocimiento limitado en el ámbito de la bioquímica en general ha condicionado la profundidad con la que pude explorar el proyecto. Contar con una base más sólida en estos campos habría permitido un aprovechamiento más completo de la herramienta, especialmente al poder probar la interfaz con un entendimiento más técnico de las propiedades de átomos y enlaces. Este factor también se refleja en el diseño e implementación del modelo GNN. Aunque la química no es un factor crítico en este contexto, dado que el modelo se centra en extraer propiedades de los átomos y enlaces para procesarlas, un conocimiento más profundo en bioquímica habría enriquecido la calidad técnica del trabajo. Además, me hubiese gustado profundizar más en este último tema e investigar diferentes formas de entrenar un modelo teniendo la limitiación de un cojunto de datos pequeño.

A pesar de estas limitaciones, considero que lo diseñado, desarrollado e implementado constituye una solución viable y efectiva para el objetivo propuesto.

9.1 Limitaciones y posibles mejoras

En cuanto a la interfaz gráfica, es cierto que la interacción podría considerarse algo rígida, o más bien, no tan fluida como me gustaría. Al final, lo que se muestra en la aplicación es simplemente una imagen SVG que se actualiza en tiempo real cada vez que hay un cambio en su estructura. Sería interesante explorar y comparar otros frameworks, fuera de Python, e investigar cómo integrar la funcionalidad de RDKit en ellos para mejorar el rendimiento y la experiencia de usuario.

Respecto a las posibles mejoras, aunque no son infinitas, existen muchas formas de enriquecer la aplicación. Por ejemplo, se podría implementar funciones adicionales de deshacer, al estilo del típico "Ctrl + Z", para corregir errores rápidos como un "click" que añade un átomo incorrecto. Además, sería útil asociar estas funciones con atajos de teclado para mejorar la usabilidad. Más allá de esto, se podría profundizar en las funcionalidades que ofrece RDKit y, a partir de ahí, crear nuevos métodos que aprovechen estas capacidades.

Finalmente, la interfaz gráfica, en términos estéticos, tiene espacio para mejorar, lo que permitiría optimizar aún más la experiencia visual y de usuario.

En cuanto a los modelos GNN y sus resultados, se podría concluir que las predicciones deben tomarse con cautela, principalmente debido a la pequeña cantidad de moléculas/ligandos disponibles para el entrenamiento. Es complicado ajustar el modelo de manera que no se sobreajuste, es decir, que no aprenda los datos "demasiado bien" y la vez para que las predicciones sean precisas para componentes nunca vistos. Si el modelo se adapta excesivamente a los ligandos utilizados durante el entrenamiento, al intentar predecir algo fuera de ese conjunto de datos, la predicción generada será muy inexacta. Es crucial que el modelo sea lo suficientemente generalizado para ofrecer predicciones mínimamente correctas en moléculas no vistas anteriormente.

En cuanto a posibles mejoras, estoy convencido de que tanto el modelo como el proceso de entrenamiento pueden optimizarse. Una de las más efectivas sería el ajuste automatizado de hiperparámetros, definiendo rangos adecuados y evaluando distintas combinaciones mediante métodos como búsqueda en rejilla (grid search), búsqueda aleatoria (random search) o técnicas más avanzadas como Bayesian optimization. También sería recomendable aumentar el volumen y la diversidad del conjunto de datos, utilizar técnicas

de data augmentation específicas para grafos, o incluso explorar arquitecturas más sofisticadas adaptadas al dominio químico.

9.2 Aportaciones personales del proyecto

En cuanto a mi aportación personal en este proyecto, he tenido la oportunidad de profundizar en el uso de Python y todas sus ventajas. Al ser un lenguaje con una sintaxis tan intuitiva, me ha permitido desarrollar soluciones rápidamente, lo que facilita la creación de prototipos que, a su vez, pueden servir como base para proyectos más grandes. La creación de la interfaz gráfica, por ejemplo, me ha ayudado a comprender las dificultades que implica desarrollar este tipo de aplicaciones, como la importancia de tener un código bien estructurado y fácil de entender.

Respecto al desarrollo de las GNN, he descubierto que, además de ser relativamente sencillo de implementar gracias a las numerosas herramientas disponibles, hay una gran cantidad de información que facilita su aprendizaje y aplicación. Como mencioné en la introducción, las GNN no tienen un uso único o específico, aunque pueden ser más útiles en ciertos campos que en otros. Lo que más me sorprende y me motiva es el hecho de que las GNN tienen un abanico tan amplio de aplicaciones, lo que abre muchas posibilidades para su uso en distintas áreas.

10 Referencias

[1] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. IEEE transactions on neural networks, 20(1), 61-80.

[2] Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

[3] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. arXiv preprint arXiv:1710.10903.

[4] Ramponi, M. (2024, February 20). AI trends in 2024: Graph Neural Networks. https://www.assemblyai.com/blog/ai-trends-graph-neural-networks

[5] Página web RDKit: Open-source cheminformatics software (versión 2024.9.6) [Software]. https://www.rdkit.org [Consulta: 08/03/2025]

[6] Página web PySide6 (versión 6.9.0) [Software]. PyPI. https://pypi.org/project/PySide6/ [Consulta: 08/03/2025]

[7] Página web PyTorch (versión 2.7.0) [Software]. https://pytorch.org/ [Consulta: 05/05/2025]

[8] Página web PyTorch Geometric (versión 2.6.1) [Software]. https://pyg.org/ [Consulta: 05/05/2025]

[9] Página web Scikit-learn (versión 1.6.1) [Software]. https://scikit-learn.org/stable/ [Consulta: 07/05/2025]

[10] Página web https://marketplace.visualstudio.com/items?itemName=d-koppenhagen.file-tree-to-text-generator [Consulta: 21/05/2025]

[11] Página web https://lifechemicals.com/order-and-supply/how-to-work-with-sd-files [Consulta: 8/03/2025]

[12] Página web https://www.pythonguis.com/tutorials/pyside6-signals-slots-events/ [Consulta: 12/03/2025]

[13] Página web NumPy (versión 2.2.5) [Software]. https://numpy.org/ [Consulta: 25/03/2025]

[14] Página web Jaime. (2013). Finding the closest point to a list of points. Stack Overflow. https://codereview.stackexchange.com/questions/28207/finding-the-closest-point-to-a-list-of-points [Consulta: 20/03/2025]

[15] Página web Jannik, E. (2019). https://www.cheminformania.com/rdeditor-an-open-source-molecular-editor-based-using-python-pyside2-and-rdkit/#:~:text=The%20conversion%20of,returned%20SVG%20coordinates. [Consulta: 25/03/2025]

[16] Página web Faik, L. (2022). https://blog.dataiku.com/graph-neural-networks-part-three#:~:text=What%20are%20the%20node%20features%3F [Consulta:05/05/2025]

[17] Página web. https://stats.stackexchange.com/questions/563763/nodes-attribute-scaling-normalization-before-graph-embedding-learning-gnn [Consulta: 06/05/2025]

[18] Página web. https://www.digitalocean.com/community/tutorials/standardscaler-function-in-python [Consulta: 06/05/2025]

[19] Página web. https://www.ibm.com/es-es/think/topics/convolutional-neural-networks [Consulta: 06/05/2025]

[20] Página web Klinger, N. (2024). https://viso.ai/deep-learning/batch-normalization/ [Consulta: 06/05/2025]

[21] Página web. https://stackoverflow.com/questions/42070528/what-does-global-pooling-do [Consulta: 06/05/2025]

[22] Página web. https://www.dremio.com/wiki/dropout-in-neural-networks/ [Consulta: 06/05/2025]

[23] Página web Pykes, K. (2024). https://www.datacamp.com/es/tutorial/adamw-optimizer-in-pytorch [Consulta: 06/05/2025]

[24] Página web. https://wiki.cloudfactory.com/docs/mp-wiki/scheduler/reducelronplateau [Consulta: 06/05/2025]

[25] Página web. https://labex.io/tutorials/python-early-stopping-for-machine-learning-300214 [Consulta: 06/05/2025]

[26] Página web. https://stackoverflow.com/questions/72274562/mseloss-from-pytorch [Consulta: 06/05/2025]

11 Anexos

11.1 Guía de instalación

IMPORTANTE: Es necesario tener la última versión de python 3.13.3 instalada para que las dependencias no den ningún error.

0. Descargar el proyecto

   Antes de comenzar, descarga el código fuente del proyecto desde el repositorio oficial:

   - Clonar con Git:

     git clone https://github.com/lakambra/quimic-editor.git

     cd quimic-editor

   - O descargar ZIP:

     Accede a https://github.com/lakambra/quimic-editor

     Haz clic en el botón Code → Download ZIP

     Extrae el contenido en una carpeta y abre una terminal en esa ubicación.

1. Crear y activar un entorno virtual
   - En la raíz del proyecto, abre una terminal y ejecuta:
     - En windows:

       python -m venv .venv

       .\.venv\Scripts\activate

     - En macOS/Linux:

       python3 -m venv .venv

       source .venv/bin/activate

2. Instalar las dependencias
   - Ejecuta:

     pip install --upgrade pip

     pip install -r requirements.txt

3. Inicia el proyecto desde la raiz
   - Ejecuta:

     python main.py