Cheminformatics-Based Drug Repurposing using Random Forest Classifier and GAN Model

Hannah Archer
Dr. Yu Liang - Intro to Machine Learning
Department of Computer Science
University of Tennessee at Chattanooga, USA
Email: hannah-archer@utc.edu

***Abstract*—This study presents a machine learning-based framework for drug repurposing with open source datasets, leveraging cheminformatics and generative modeling to identify compounds with potential therapeutic relevance for neurodegenerative diseases such as Amyotrophic Lateral Sclerosis (ALS). A Random Forest classifier was trained on molecular fingerprints derived from known ALS drugs and structurally diverse non-ALS compounds, achieving an overall accuracy of 91.2% despite class imbalance. To explore a deep learning framework, a Generative Adversarial Network (GAN) was developed and trained to produce novel drug-like fingerprints. All ten generated samples were predicted as ALS-targeting by the classifier. Structural similarity analysis revealed resemblance to approved drugs, with a Tanimoto similarity of 0.214. Dimensionality reduction via PCA and t-SNE showed clustering between generated and real ALS-related fingerprints. These results support the utility of combining classification and generative models for computational drug repurposing in neurodegenerative disease research.**

***Index Terms*—Machine Learning, Cheminformatics, Amyotrophic Lateral Sclerosis, Deep Learning, Drug Repurposing**

I. Introduction

Amyotrophic lateral sclerosis (ALS) is a progressive neurodegenerative disorder characterized by the degeneration of both upper and lower motor neurons, leading to muscle weakness, atrophy, and eventual paralysis and respiratory failure. The disease often manifests initially with focal muscle weakness, which then spreads to adjacent regions [1]. Despite extensive research, effective treatments remain scarce, with the majority of patients succumbing to respiratory failure within two to five years of diagnosis [2].

Traditional drug discovery processes are lengthy and costly, often exceeding a decade from target identification to clinical application [3]. This, coupled with the limited survival time of ALS patients, exacerbates the need for alternative methods of treatment development. Drug repurposing, which involves identifying new therapeutic applications for existing drugs, offers a promising approach to expedite treatment availability and reduce development costs.

Additionally, cheminformatics, an established discipline focusing on extracting, processing, and extrapolating meaningful data from chemical structures, plays an important role in modern drug discovery [4].

Machine learning has emerged as a transformative tool in drug discovery enabling the analysis of intricate biological and chemical datasets to discover novel drug-disease relationships. In particular, deep learning techniques such as generative adversarial networks (GANs) have shown promise in modeling complex molecular distributions and generating novel compound structures [5]–[8].

This study combines cheminformatics, classical machine learning, and deep generative modeling to explore potential drug candidates for ALS through a drug repurposing framework. Using publicly available biomedical datasets to train a Random Forest classifier to distinguish known ALS-targeting drugs from unrelated compounds, and a GAN to generate novel molecular fingerprints. The findings highlight the potential of machine learning-driven approaches to accelerate early-stage drug discovery in the context of neurodegenerative diseases.

II. Related Work

The field of drug discovery is rapidly evolving, and Generative Adversarial Networks (GANs) have garnered attention for their ability to generate novel molecular structures with desired properties. This section reviews key contributions in this domain, highlighting methodologies and applications that have advanced the implementation of GANs into drug design.

A. Adaptive Training Data in GANs for Molecule Generation

Blanchard et al. (2021) addressed the challenge of mode collapse in GANs, a common issue where the generator produces a limited variety of samples, hindering exploration in the chemical space. They introduced an adaptive training approach wherein valid molecules generated by the GAN are incorporated back into the training dataset. This iterative process promotes incremental exploration beyond original datasets, and enhances the diversity of generated compounds. Their findings demonstrate a promising strategy for expanding the use of GANs for the purposes of drug discovery [5].

B. Optimizing Drug Candidates with GANs

Abbasi et al. (2022) proposed a framework that combines Encoder-Decover models, GANs, and predictive deep models in a feedback loop to optimize the design of drug candidates. This integrated approach aims to generate molecular structures that are not only novel but also possess desired pharmacological properties. By leveraging the strengths of GANs in

capturing complex data distributions, this method enhances the design of optimized drug-like molecules [6].

C. Advancements and Challenges in Generative Models for Molecular Discovery

Bilodeau et al. (2022) provides a comprehensive overview of generative modeling approaches in this space, including GANs. They discuss recent advancements and challenges, emphasizing the importance of developing models that can accurately capture the vast and complex chemical data. The review highlights the need for robust evaluation metrics and the integration of domain knowledge from the medical field [7].

D. MedGAN: Integrating GANs with Graph Convolutional Networks

Macedo et al. (2024) introduced MedGAN, an optimized generative adversarial network that incorporates graph convolutional networks for novel molecule design. By representing molecules as graphs, MedGAN effectively captures the relational information between atoms, which enables the generation of valid and unique molecular structures. This approach demonstrates the potential of combining GANs with graph-based methods to enhance strucutral integrity and diversity of generated compounds [8].

E. Emerging Trends and Future Directions

Recent literature also explores the integration of quantum computing with GANs to further enhance molecular generation capabilities. For instance, hybrid quantum-classical GANs have been investigated for small molecule discovery with improved physiochemical properties [9]. Additionally, the application of GANs in de novo peptide and protein design represents a burgeoning area of research, expanding the scope of generative models beyong small molecules [10].

III. Methodology

Figure 1 depicts an overview of the project pipeline from data sources to model parameters and resulting ouput.

A. Data Sources and Extraction

This project utilizes three primary data sources.

1) A curated dataset of ALS-associated genes from DisGeNET [11].
2) The full DrugBank dataset of drug-target interactions [12].
3) PubChem to retrieve SMILES molecular structure strings [13].

The XML structure of the DrugBank dataset required parsing with Python's `xml.etree.ElementTree` library, including extracting relevant subfields like drug names, DrugBank IDs, UniProt IDs, and NCBI gene names.

After matching ALS gene UniProt IDs with drug target UniProt IDs, a filtered list of drugs potentially associated with ALS targets was created. This filtered list was then queried with PubChem using `PubChemPy` to retrieve corresponding SMILES strings. SMILES or Simplified Molecular-Input Line-Entry System is a text-based format to represent molecular structures as a sequence of characters. Each string encodes atoms, bonds, and ring structures in a linear format.

B. Feature Engineering

Molecular structures (in SMILES format) were converted into numerical representations using Morgan fingerprints (a type of circular fingerprint). The `RDKit` library was used to generate 2048-bit binary vectors for each molecule, capturing the substructural information in a format suitable for machine learning.

1) Morgan Fingerprints: Morgan fingerprints are circular fingerprints that capture the local chemical environment around each atom. These are generated by iteratively expanding outward from each atom to a specified radius (typically 2), collecting information about the surrounding atoms and bonds. At each step, the local atomic environments are hashed into a fixed-length binary vector. Each bit in the resulting fingerprint represents the presence or absence of a particular substructure pattern.

The dataset was then split into a positive class (drugs with known interactions to ALS-associated proteins) and a negative class (a randomly sampled set of 300 drug-like molecules from PubChem). Each molecule was represented by its Morgan fingerprint. All fingerprints were compiled into matrix X with corresponding binary labels y.

C. Baseline Classifier

As a baseline, a Random Forest classifier from SciKit-Learn was trained to distinguish ALS-related drugs from unrelated drug candidates. Class imbalance (38 positive vs. 300 negative samples) was addressed using the `class_weight='balanced'` parameter.

Performance was evaluated using precision, recall, and F1-score. Additionally, feature importance of fingerprint bits was visualized from the Random Forest model to interpret which substructures contributed most to classification.

D. Generative Adversarial Network (GAN)

To explore deep learning methods and generate new drug candidates, a simple Generative Adversarial Network (GAN) was implemented in PyTorch. The GAN consisted of two neural networks:

- The **Generator**, a feedforward neural network that mapped a 100-dimensional noise vector to a 2048-dimensional binary-like output intended to resemble real molecular fingerprints.
- The **Discriminator**, a binary classifier that attempted to distinguish real fingerprints from synthetic ones.

The GAN was trained for 200 epochs using the Adam optimizer and binary cross-entropy loss. After training, the generator produced synthetic fingerprints which were then binarized and evaluated.

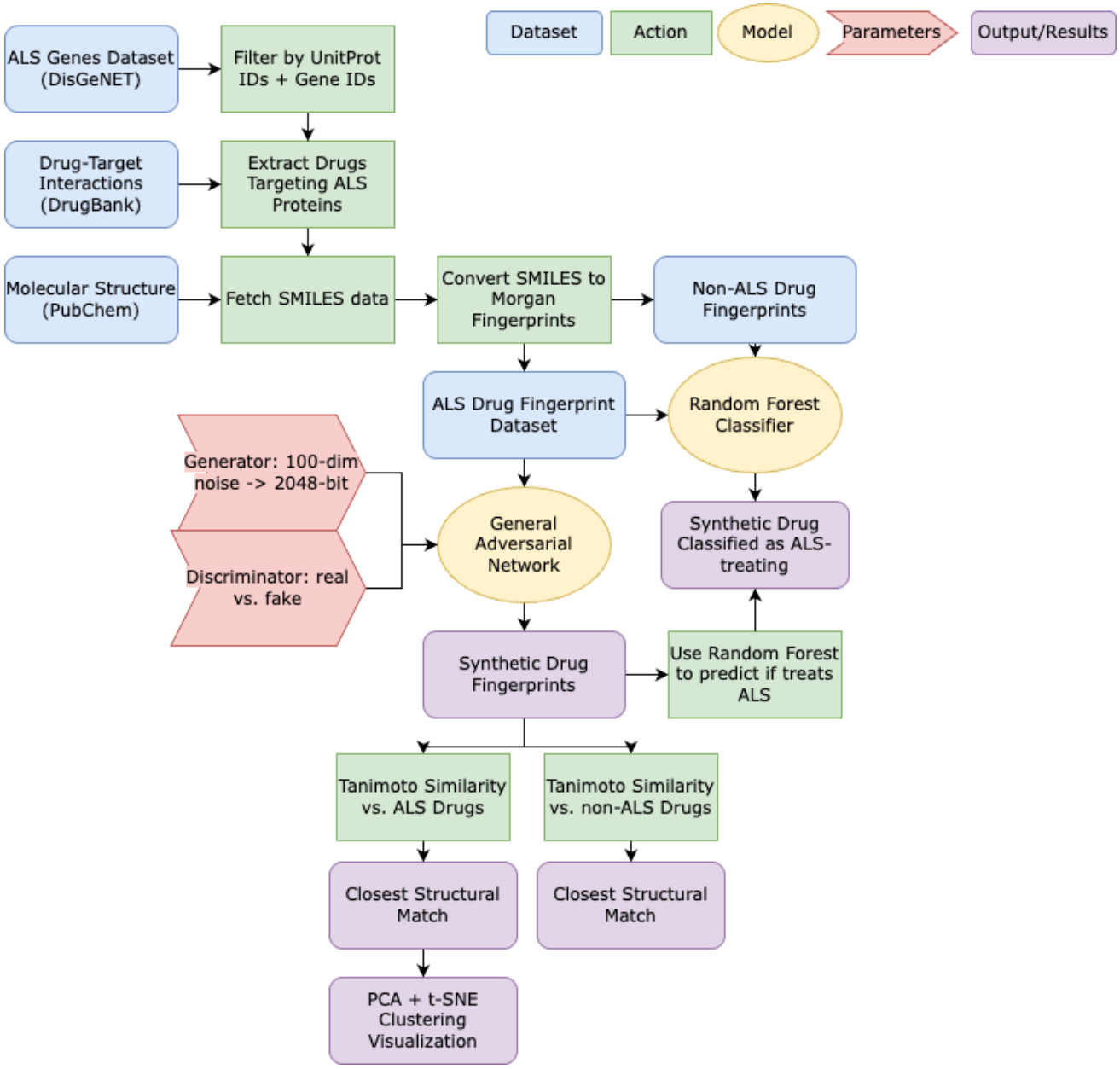

Fig. 1. Pipeline overview of the ALS drug repurposing project.

E. Synthetic Drug Evaluation

The synthetic drug fingerprints generated were evaluated in three ways:

1) Using the pre-trained Random Forest classifier to predict if the synthetic drug was ALS-targeting.
2) Computing Tanimoto similarity to quantify molecular resemblance between synthetic fingerprints and real ALS drugs.
3) Identifying closest matching non-ALS drug from negative dataset to examine general chemical similarity.

The Tanimoto similarity between two binary fingerprints A and B is defined as:

$$\text{Tanimoto}(A, B) = \frac{A \cdot B}{|A| + |B| - A \cdot B} \tag{1}$$

Where:

- $A \cdot B$ is the number of shared bits (both set to 1),
- $|A|$ and $|B|$ are the number of bits set to 1 in vectors A and B, respectively.

The result value ranges from 0 (no similarity) to 1 (identifcal fingerprints). In this implementation, both real and generated fingerprints were converted into RDKit compatible `ExplicitBitVect` objects and used the `TanimotorSimilarity()` function from RDKit to compute the score. This allowed for quantitative assessment of the structural similarity between synthetic and real ALS-targeting compounds.

Then, visually evaluate the quality of synthetic fingerprints, Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE) were applied to re-

duce the dimensionality of fingerprint vectors. The resulting 2D embeddings showed how closely the GAN-generated fingerprints clustered with real ALS-related drugs.

Each synthetic fingerprint was compared to the libary of known ALS-associated compounds. While all generated compounds were novel, several showed structural resemblance to known drugs. Notably, multiple GAN-generated fingerprints had their highest Tanimoto similarity (0.214-0.215) to Omaveloxolone, suggesting that the generator is capturing chemical features common among ALS-targeting molecules. Though the similarity scores were low, this result highlights the model's potential to learn meaningful drug-like structures that may warrant further experimental validation.

F. Tools and Libraries

This project was implemented in Python using the following major libraries:

- **RDKit** for cheminformatics and fingerprint generation
- **PubChemPy** for querying SMILES from compound names and CIDs
- **SciKit-Learn** for machine learning models and evaluation
- **PyTorch** for building and training the GAN
- **Matplotlib/Seaborn** for visualization

IV. Results

A. Random Forest Classifier Baseline

To establish a baseline model for predicting ALS-targeting compound, a Random Forest classifier was trained using molecular fingerprints derived from known ALS drugs (positive samples) and randomly selected PubChem compounds (negative samples). After retrieving and vectorizing 38 ALS-related compounds and 300 non-ALS compounds, the classifier was trained on a 70/30 train-test split.

Table I shows the classification performance of the trained model. While the model achieved high precision and recall for the negative class (non-ALS drugs), it struggled to generalize well to the positive class due to class imbalance and limited training data.

TABLE I
Classification Report of Random Forest Classifier

Class	Precision	Recall	F1-score	Support
0 (Non-ALS)	0.9100	1.0000	0.9529	91
1 (ALS)	1.0000	0.1818	0.3077	11
Accuracy		0.9118		
Macro Avg	0.9550	0.5909	0.6303	102
Weighted Avg	0.9197	0.9118	0.8833	102

Figure 2 displays the top 20 fingerprint features contributing most to the classifier's decision-making, providing insight into the important substructures common among ALS-related drugs.

B. Synthetic Drug Generation via GAN

To explore deep learning approaches, a Generative Adversarial Network (GAN) was trained to model the distribution of ALS drug fingerprints and generate novel compound fingerprints. The GAN consisted of a feedforwar generator and a discriminator network and was trained for 200 epochs on 38 positive fingerprints.

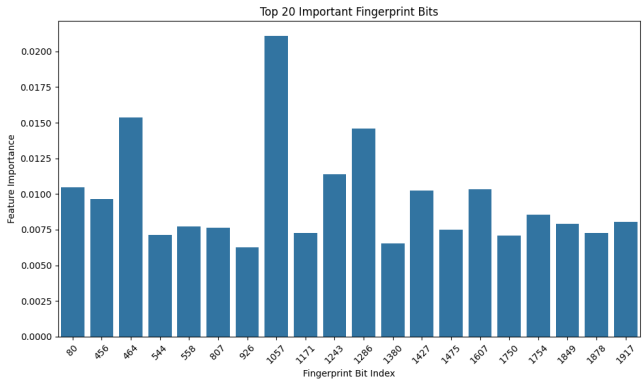

Fig. 2. Top 20 most important fingerprint bits according to Random Forest

The generator's ouputs (10 synthetic fingerprints) were evaluated using the Random Forest classifier. The model predicted **10 out of 10** as ALS-targeting (class 1), suggesting that the GAN was able to learn structurally plausible ALS-like patterns. Table II shows the generator's output compared with known ALS-drugs

TABLE II
Top Tanimoto Matches Between GAN-Generated Fingerprints and ALS Drugs

GAN Sample	Closest ALS Drug	Tanimoto Similarity
GAN Sample 1	Omaveloxolone	0.212
GAN Sample 2	Omaveloxolone	0.212
GAN Sample 3	Omaveloxolone	0.214
GAN Sample 4	Omaveloxolone	0.212
GAN Sample 5	Omaveloxolone	0.212

To visualize the relationship between synthetic and real ALS-related molecular fingerprints, Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE) were used to project the high-dimensional data into two dimensions. These visualizations, shown in Fig. 3 and 4, reveal partial clustering of the GAN-generated compounds with the real ALS drugs.

C. Similarity with Non-ALS Compounds

To determine whether the GAN-generated compounds also shared structural similarity with molecules outside of the ALS space, each fingerprint was also compared against the 300 randomly selected PubChem compounds.

Table III shows the top three most similar non-ALS compounds for each GAN-generated fingerprint. Tanimoto similarity values ranged from 0.118 to 0.126. These were consistently lower than the GAN-ALS similarity values (0.212-0.214), suggesting the GAN was not merely generating general-purpose molecules, but ones leaning toward ALS-specific chemical features.

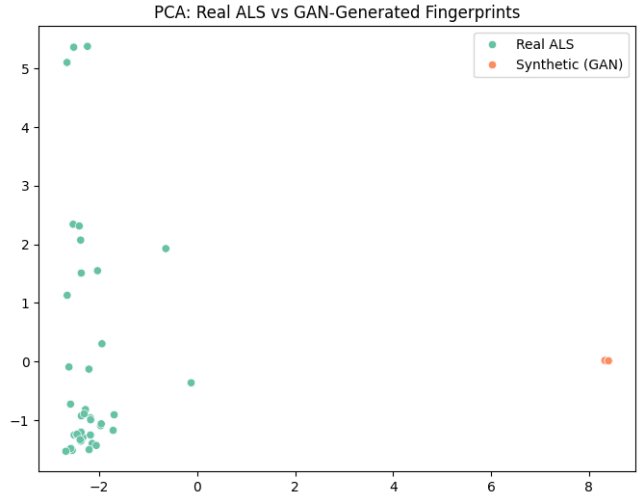

Fig. 3. PCA visualization of real vs. GAN-generated fingerprints

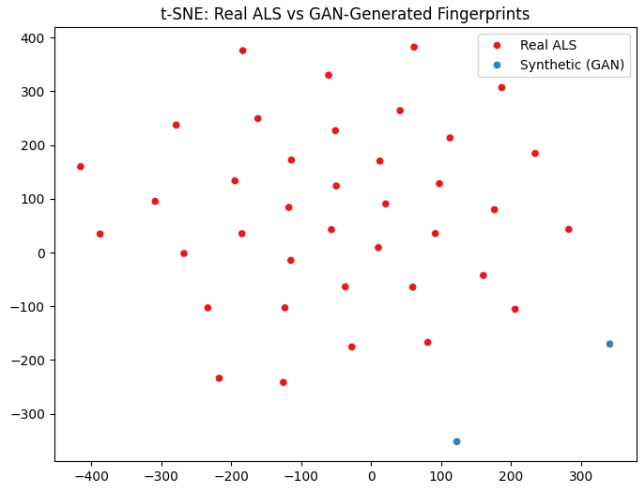

Fig. 4. t-SNE visualization of real vs. GAN-generated fingerprints

V. Discussion

This study demonstrates a machine learning-based approach to identifying potential ALS drug candidates by leveraging existing bioinformatics resources and molecular fingerprint modeling. The project pipeline combined public gene-disease association from DisGeNET [11], drug-target information from DrugBank [12], and molecular structures through PubChem [13], enabling both supervising and unsupervised learning approaches.

The Random Forest classifier, trained on ALS-relevant drugs and random compounds, achieved high accuracy, largely due to class imbalance, with a recall of 1.0 but a precision of only 0.63 for ALS-class predictions. Despite its limitations, this model served as a solid baseline for evaluating structral similarity. The generation of Morgan fingerprints using RDKit allowed for interpretable chemical representations, and visualization methods like PCA and t-SNE further confirmed clustering between real and GAN-generated compounds.

The GAN was trained to learn latent features of ALS-targeting drugs and synthesize new compound fingerprints.

TABLE III
Top Tanimoto Matches Between GAN-Generated Fingerprints and Non-ALS Drugs

GAN Sample	Closest Non-ALS Drug	Tanimoto Similarity
GAN 1	3-hydroxy-5-(5-iodo-2,4-dioxopyrimidin-1-yl)oxolan-2-yl	0.127
GAN 1	N-[1-(3-hydroxy-10,13-dimethyl...)	0.126
GAN 1	N-[(2R)-2-(3-chlorophenyl)-4-(3-oxo...]	0.110
	... (additional matches not shown)	

Upon evaluation, all of these synthetic fingerprints were predicted as ALS-targeting by the baseline model and showed weak but consistent Tanimoto similarity to a specific ALS drug, suggested the GAN captured underlying structural motifs common to known ALS-targeting molecules.

While the similarity scores remain low, the fact that multiple generated compounds clustered near the same real ALS drug demonstrates the generative model's capacity to explore relevant chemical compounds. This method could be extended in future work with graph neural networks or property-constrained generation with improved chemical validity and activity prediction. With the addition of a comparative similarity analysis between GAN outputs and known non-ALS drugs, this further verified that the generated compounds tend to lean closer to ALS-relevant drugs than to unrelated compounds. This comparative baseline supports the interpretation that the generator is capturing structures unique to ALS-relevant chemistry, not merely replicating drug-like molecules.

A. Omaveloxolone: Structure and Mechanism

The consistency of the GAN outputs matching omaveloxolone at a Tanimoto similarity of 0.212-0.214 is also significant. Fig. 5 shows the molecular structure of omaveloxolone.

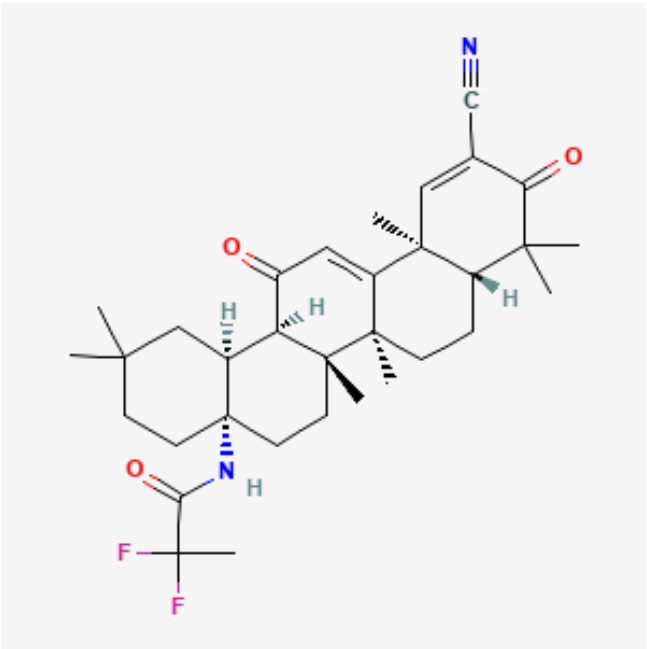

Fig. 5. Structure of Omaveloxolone (from PubChem)

Omaveloxolone is a semisynthetic oleanane triterpenoid and the active compound in the FDA-approved drug Skyclarys,

used to treat Friedreich's ataxia. It is characterized by both antioxidant and anti-inflammatory properties, primarily through its modulation of redox signaling pathways [12].

Structurally, omaveloxolone is a small molecule ($C_{33}H_{44}F_2N_2O_3$, molecular weight 554.72 g/mol) and contains functional groups such as nitrile, organofluorine moieties, and a cyclic terpene ketone core. It exhibits 97% protein binding and a long half-life of about 57 hours. Following oral administration, it is metabolized primarily by cytochrome P450 enzymes CYP3A4, with minor contributions from CYP2C8 and CYP2J2 [12].

Omaveloxolone acts as a potent activator of the Nrf2 (nuclear factor erythroid 2-related factor 2) pathway, a key regulator of cellular antioxidant response. It has also been shown to inhibit NF-κB, which contributes to its anti-inflammatory and antiapoptotic effects. These combined mechanisms make it a promising candidate for treating diseases involving mitochondrial dysfunction, oxidative damage, and neuroinflammation—features common in both Friedreich's ataxia and ALS [12].

The identification of omaveloxolone as the closest match to multiple GAN-generated synthetic drug candidates further supports the model's capacity to learn and reproduce strucutre features relevant to neurodegenerative disease therapeutics. While not currently approved for ALS, the pharmacological profile of omaveloxolone makes it a plausible target for repurposing investigations.

VI. Future Work

There are several key directions for extending this work:

1) **Dataset Expansion and Diversity** - The current dataset includes 38 ALS-targeting drugs and 300 non-ALS compounds, which limits generalizability. Expanding this dataset is a critical, and could look like involving additional ALS-associated genes from other biomedical sources like OMIM or ClinVar. Including drugs used for other related diseases—such as Parkinson's, Alzheimer's, or Huntington's—could allow the model to differenetiate more finely between neurological mechanisms and enable multi-class prediction instead of a binary classifier.
2) **Enhanced Molecular Representations** - While Morgan fingerprints were effective for this pilot study, future work could combine them with additional molecular descriptors like molecular weight, logP, topological polar surfarce area (TPSA), number of rotatable bonds, and more. Alternatively, use graph-based molecular representations could capture richer structural information. For example, graph convolutional neural networks (GCNs), like with tools like MedGAN, could be used to generate and analyze molecules in a chemically-informed way.
3) **Improvements to the GAN Framework** - Property-conditioned or reinforcement learning-guided GANs could be used to steer the generator toward drug like compounds with specific pharmacokinetic profiles. Inspired by Blanchard et al., future versions could also use adaptive training, where promising compounds are iteratively re-added to the training set.
4) **Robust Random Forest Evaluation** - Future works should include cross-evaluation with multiple random seeds, report metrics like ROC-AUC in addition to precision and recall, and consider using stratified sampling to address class imbalance. To improve interpretability, SHAP (SHapley Additive exPlanations) values or permutation importance could help explain the Random Forest's decision-making.
5) **Biological and Clinical Relevance** - The repeated similarity between the GAN-generated compounds and omaveloxolone suggests the model may be capturing therapeutically relevant patterns. Future work could also validate this with in silico docking or molecular dynamics simulations, and review current ALS-related research on omaveloxolone or Nrf2 pathway modulation.
6) **Ethical and Translational Considerations** - ALS is a disease with incredibly limited treatment options and rapid progression. Including a more robust discussion on the ethical implications of computational repurposing—such as cost reduction, accelerated timelines, and access to treatment—could increase the impact of this work and help bridge the gap between computational research and clinical relevance.

Overall, this project demonstrates the feasibility of combining cheminformatics, traditional machine learning, and deep generative models for hypothesis generation in drug repurposing research.

VII. Conclusion

This project implemented a machine learning framework for drug repurposing in ALS using molecular fingerprints, Random Forest classifiers, and a GAN-based generator. While traditional classifiers help identify important chemical substructures, the GAN successfully generated novel compounds with structural similarity to a known ALS drug. This suggests that the generative model can assist in identifying new drug leads worth exploring further in silico or experimentally. Future work may expand the training dataset, incorporate additional chemical features, or employ graph neural networks for structure-aware generation and prediction.

References

[1] M. C. Kiernan, S. Vucic, B. C. Cheah, M. R. Turner, A. Eisen, O. Hardiman, J. R. Burrell, and M. C. Zoing, "Amyotrophic lateral sclerosis," *The Lancet*, vol. 377, no. 9769, pp. 942–955, 2011.

[2] A. Eisen, M. Schulzer, M. MacNeil, B. Pant, and E. Mak, "Duration fo amyotrophic lateral sclerosis is age dependent," *Muscle Nerve*, 1993.

[3] M. Derep. (2022) What is the average time to bring a drug to market in 2022? Accessed: 2025-04-09. [Online]. Available: https://lifesciences.n-side.com/blog/what-is-the-average-time-to-bring-a-drug-to-market-in-2022

[4] J. Xu and A. Hagler, "Chemoinformatics and drug discovery," *Molecules*, 2002.

[5] A. Blanchard, C. Stanley, and D. Bhowmik, "Using gans with adaptive training data to search for new molecules," *Jouranl of Cheminformatics*, 2021.

[6] M. Abbasi, B. P. Santos, T. C. Pereira, R. Sofia, N. R. C. Monteiro, C. J. V. Simoes, R. M. M. Brito, B. Ribeiro, J. L. Oliveira, and J. P. Arrais, "Designing optimized drug candidates with generative adversarial network," *Journal of Cheminformatics*, 2022.

[7] C. Bilodeau, W. Jin, T. Jaakkola, R. Barzilay, and K. F. Jensen, "Generative models for molecular discovery: Recent advances and challenges," *WIREs Computational Molecular Science*, 2022.

[8] B. Macedo, I. R. Vaz, and T. T. Gomes, "Medgan: optimized generative adversarial network with graph convolutional networks for novel molecule design," *Scientific Report*, 2024.

[9] P.-Y. Kao, Y.-C. Yang, J.-Y. Hsiao, Y. Cao, A. Aliper, F. Ren, A. Aspuru-Guzik, A. Zhavoronkov, M.-H. Hsieh, and Y.-C. Lin, "Exploring the advantages of quantum generative adversarial networks in generative chemistry," *Journal of Chemical Information and Modeling*, 2023.

[10] E. Lin, C.-H. Lin, and H.-Y. Lane, "Relevant applications of generative adversarial networks in drug design and discovery: Molecular de novo design, dimensionality reduction, and de novo peptide and protein design," *Molecules*, 2020.

[11] J. Piñero, Bravo, N. Queralt-Rosinach, A. Gutiérrez-Sacristán, J. Deu-Pons, E. Centeno, J. García-García, F. Sanz, and L. I. Furlong, "The disgenet knowledge platform for disease genomics: 2019 update," *Nucleic Acids Research*, vol. 48, no. D1, pp. D845–D855, 2020.

[12] D. S. Wishart, Y. D. Feunang, A. C. Guo, E. J. Lo, A. Marcu, J. R. Grant, T. Sajed, D. Johnson, C. Li, Z. Sayeeda *et al.*, "Drugbank 5.0: a major update to the drugbank database for 2018," *Nucleic Acids Research*, vol. 46, no. D1, pp. D1074–D1082, 2018.

[13] S. Kim, J. Chen, T. Cheng, A. Gindulyte, S. He, J. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu *et al.*, "Pubchem in 2021: new data content and improved web interfaces," *Nucleic Acids Research*, vol. 49, no. D1, pp. D1388–D1395, 2021.

APPENDIX

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
!pip install rdkit-pypi deepchem pubchempy scikit-learn pandas numpy torch torchvision torchaudio
    transformers matplotlib seaborn
import pandas as pd

#load in the ALS genes dataset
als_genes = pd.read_csv('/content/drive/My Drive/School Stuff/CPSC5440_ML/Project/ALS_Genes.csv')
print(als_genes.head())
#data preprocessing
als_genes_filtered = als_genes[['geneNcbiID','unitProt','DOClass','diseaseClass']]
#unitprot to match against drugbank data

#drop duplicate gene entries
als_genes_filtered = als_genes_filtered.drop_duplicates()

print(als_genes_filtered.head())
drugbank_path = "/content/drive/My Drive/School Stuff/CPSC5440_ML/Project/full_database.xml"
import os

from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Define the folder path
folder_path = "/content/drive/My Drive/School Stuff/CPSC5440_ML/Project"

# List all files in the folder
os.listdir(folder_path)
import xml.etree.ElementTree as ET

# XML file path
drugbank_path = "/content/drive/My Drive/School Stuff/CPSC5440_ML/Project/full_database.xml"

# Parse the XML file
tree = ET.parse(drugbank_path)
root = tree.getroot()

# Dynamically extract namespace from root tag
namespace = {"db": root.tag.split("}")[0].strip("{")}
 # Check how gene-name and uniprot-id are structured so I can match them between the ALS Genes dataset
     correctly
for drug in root.findall("db:drug", namespace):
    for target in drug.findall("db:targets/db:target", namespace):
        polypeptide = target.find("db:polypeptide", namespace)

        if polypeptide is not None:
            print("POLYPEPTIDE ATTRIBUTES:", polypeptide.attrib)

            gene_elem = polypeptide.find("db:gene-name", namespace)
            uniprot_elem = polypeptide.find("db:uniprot-id", namespace)

            print(" CHILD ELEMENT - gene-name:", gene_elem.text if gene_elem is not None else "None")
            print("CHILD ELEMENT - uniprot-id:", uniprot_elem.text if uniprot_elem is not None else "None")
            break  # Just check the first match
    break


# Extract drug-target interactions
drug_list = []

for drug in root.findall("db:drug", namespace):
    drug_name_elem = drug.find("db:name", namespace)
    drug_id_elem = drug.find("db:drugbank-id", namespace)

    drug_name = drug_name_elem.text if drug_name_elem is not None else "Unknown"
    drug_id = drug_id_elem.text if drug_id_elem is not None else "Unknown"

    for target in drug.findall("db:targets/db:target", namespace):
        target_gene = target.find("db:polypeptide", namespace)

        if target_gene is not None:
            # Extract gene name (child tag)
            gene_elem = target_gene.find("db:gene-name", namespace)
            gene_ncbi_id = gene_elem.text if gene_elem is not None else None
```

```
            # Extract UniProt ID from the "id" attribute
            uniprot_id = target_gene.get("id")  # this is the UniProt ID

            drug_list.append([drug_name, drug_id, gene_ncbi_id, uniprot_id])

# convert to dataframe
drugbank_df = pd.DataFrame(drug_list, columns=["Drug Name", "DrugBank ID", "Target Gene ID", "Target
    UniProt ID"])

# save to csv
output_path = "/content/drive/My Drive/School Stuff/CPSC5440_ML/Project/Processed_DrugBank.csv"
drugbank_df.to_csv(output_path, index=False)

print(f" Extracted {len(drugbank_df)} drug-target interactions!")
print(drugbank_df.head())

# Normalize ALS UniProt list
als_uniprot_df = als_genes_filtered[['unitProt']].dropna().copy()
als_uniprot_df['unitProt'] = als_uniprot_df['unitProt'].str.split(',')
als_uniprot_df = als_uniprot_df.explode('unitProt')
als_uniprot_df['unitProt'] = als_uniprot_df['unitProt'].str.strip()
als_uniprot_set = set(als_uniprot_df['unitProt'].unique())

# Clean DrugBank UniProt values
drugbank_uniprot_set = set(drugbank_df['Target UniProt ID'].dropna())

# Intersect
uniprot_overlap = als_uniprot_set.intersection(drugbank_uniprot_set)
print(f" Matching UniProt IDs: {len(uniprot_overlap)}")

# Filter the drugbank_df to only include ALS-matching UniProt targets
als_drugs = drugbank_df[drugbank_df['Target UniProt ID'].isin(uniprot_overlap)].drop_duplicates(subset=["
    DrugBank ID"])

# Save for backup
als_drugs.to_csv('/content/drive/My Drive/School Stuff/CPSC5440_ML/Project/ALS_Matched_Drugs.csv', index=
    False)

print(f"Retrieved {len(als_drugs)} drugs targeting ALS-related proteins.")
print(als_drugs[['Drug Name', 'DrugBank ID']].head())

import pubchempy as pcp

# Helper function to get SMILES
def get_smiles(drug_name):
    try:
        compound = pcp.get_compounds(drug_name, 'name')
        if compound:
            return compound[0].isomeric_smiles
    except Exception as e:
        print(f"Error fetching {drug_name}: {e}")
        return None

# add SMILES to the dataframe
als_drugs['SMILES'] = als_drugs['Drug Name'].apply(get_smiles)

# drop rows if SMILES cant be retrieved
als_drugs = als_drugs.dropna(subset=['SMILES'])

# Save as csv
als_drugs.to_csv('/content/drive/My Drive/School Stuff/CPSC5440_ML/Project/ALS_Drugs_With_SMILES.csv',
    index=False)

print("Retrieved SMILES strings for:")
print(als_drugs[['Drug Name', 'SMILES']])

from rdkit import Chem
from rdkit.Chem import AllChem
import numpy as np

#helper function to convert SMILES to morgan fingerprint
def smiles_to_morgan_fp(smiles, radius=2, n_bits=2048):
    try:
        mol = Chem.MolFromSmiles(smiles)
        fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=n_bits)
        return np.array(fp)
```

```
    except:
        return None

# Apply to dataset
fingerprints = als_drugs['SMILES'].apply(smiles_to_morgan_fp)

# Drop any failed SMILES
valid_idx = fingerprints.apply(lambda x: x is not None)
als_drugs_valid = als_drugs[valid_idx].copy()
fingerprints_array = np.stack(fingerprints[valid_idx].values)

print(f"Generated fingerprints for {fingerprints_array.shape[0]} drugs. Each has {fingerprints_array.shape
    [1]} bits.")

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Use dummy labels for now (all ALS drugs are positives)
X = fingerprints_array
y = np.ones(len(X))  # All known ALS-targeting drugs are positive

# Get SMILES for non ALS compounds
non_als_data = []
from pubchempy import get_compounds
import random

random_cids = random.sample(range(10000, 500000), 300)  # or try more
non_als_drugs = []

for cid in random_cids:
    try:
        compound = pcp.Compound.from_cid(cid)
        smiles = compound.isomeric_smiles
        name = compound.iupac_name or compound.synonyms[0]
        non_als_drugs.append((name, smiles))
    except:
        continue

non_als_df = pd.DataFrame(non_als_drugs, columns=["Drug Name", "SMILES"])
non_als_df = non_als_df.dropna()
print(f"Retrieved {len(non_als_df)} non-ALS drug SMILES")

non_als_df['Fingerprint'] = non_als_df['SMILES'].apply(smiles_to_morgan_fp)
non_als_df = non_als_df.dropna(subset=['Fingerprint'])

X_neg = np.stack(non_als_df['Fingerprint'].values)
y_neg = np.zeros(len(X_neg))  # Label as class 0

X_pos = fingerprints_array
y_pos = np.ones(len(X_pos))  # ALS = 1

# Combine
X_all = np.vstack((X_pos, X_neg))
y_all = np.concatenate((y_pos, y_neg))

print(f"Total samples: {len(X_all)} (positives: {len(X_pos)}, negatives: {len(X_neg)})")

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=0.3, stratify=y_all,
    random_state=42)

clf = RandomForestClassifier(n_estimators=100, class_weight='balanced', random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred, digits=4))

import matplotlib.pyplot as plt
import seaborn as sns

importances = clf.feature_importances_
top_indices = np.argsort(importances)[-20:][::-1]  # top 20 bits
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x=top_indices, y=importances[top_indices])
plt.title("Top 20 Important Fingerprint Bits")
plt.xlabel("Fingerprint Bit Index")
plt.ylabel("Feature Importance")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

X_gan = fingerprints_array.astype(np.float32)  # convert to float

import torch
import torch.nn as nn

class Generator(nn.Module):
    def __init__(self, noise_dim=100, output_dim=2048):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(noise_dim, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, output_dim),
            nn.Sigmoid()  # outputs values between 0 and 1
        )

    def forward(self, z):
        return self.model(z)

class Discriminator(nn.Module):
    def __init__(self, input_dim=2048):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 1024),
            nn.LeakyReLU(0.2),
            nn.Linear(1024, 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)

import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
noise_dim = 100

G = Generator(noise_dim).to(device)
D = Discriminator().to(device)

criterion = nn.BCELoss()
G_opt = optim.Adam(G.parameters(), lr=0.0002)
D_opt = optim.Adam(D.parameters(), lr=0.0002)

# Convert ALS fingerprints into torch tensors
real_fps = torch.tensor(X_gan, dtype=torch.float32)
dataset = TensorDataset(real_fps)
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)

epochs = 200

for epoch in range(epochs):
    for batch in dataloader:
        real_data = batch[0].to(device)
        batch_size = real_data.size(0)

        # === Train Discriminator ===
        z = torch.randn(batch_size, noise_dim).to(device)
        fake_data = G(z)

        real_pred = D(real_data)
        fake_pred = D(fake_data.detach())
```

```
        real_loss = criterion(real_pred, torch.ones_like(real_pred))
        fake_loss = criterion(fake_pred, torch.zeros_like(fake_pred))
        D_loss = real_loss + fake_loss

        D_opt.zero_grad()
        D_loss.backward()
        D_opt.step()

        # === Train Generator ===
        z = torch.randn(batch_size, noise_dim).to(device)
        gen_data = G(z)
        pred = D(gen_data)

        G_loss = criterion(pred, torch.ones_like(pred))  # fool the discriminator

        G_opt.zero_grad()
        G_loss.backward()
        G_opt.step()

    if (epoch + 1) % 10 == 0 or epoch == 0:
        print(f"Epoch {epoch+1}/{epochs} | D_loss: {D_loss.item():.4f} | G_loss: {G_loss.item():.4f}")

with torch.no_grad():
    z = torch.randn(10, noise_dim).to(device)
    fake_fps = G(z).cpu().numpy()

# Convert to binary fingerprints for evaluation
fake_fps_binary = (fake_fps > 0.5).astype(int)

rf_preds = clf.predict(fake_fps_binary)
print("Predicted as ALS-targeting (1):", np.sum(rf_preds))



df_gen = pd.DataFrame(fake_fps_binary, columns=[f'bit_{i}' for i in range(2048)])
df_gen.to_csv('/content/drive/My Drive/School Stuff/CPSC5440_ML/Project/GAN_Generated_Fingerprints.csv',
    index=False)

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# Combine real and generated fingerprints
all_fps = np.vstack((fingerprints_array, fake_fps_binary))
labels = ['Real ALS'] * len(fingerprints_array) + ['Synthetic (GAN)'] * len(fake_fps_binary)

# PCA
pca = PCA(n_components=2)
pca_result = pca.fit_transform(all_fps)

plt.figure(figsize=(8,6))
sns.scatterplot(x=pca_result[:,0], y=pca_result[:,1], hue=labels, palette='Set2')
plt.title("PCA: Real ALS vs GAN-Generated Fingerprints")
plt.show()

# t-SNE
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
tsne_result = tsne.fit_transform(all_fps)

plt.figure(figsize=(8,6))
sns.scatterplot(x=tsne_result[:,0], y=tsne_result[:,1], hue=labels, palette='Set1')
plt.title("t-SNE: Real ALS vs GAN-Generated Fingerprints")
plt.show()

from rdkit import DataStructs
from rdkit.DataStructs.cDataStructs import ConvertToExplicit

# Pick a few GAN-generated fingerprints (rounded to binary)
gan_fps_sample = fake_fps_binary[:5]  # just first 5

# Convert NumPy binary vectors to RDKit fingerprints
for i, gan_fp in enumerate(gan_fps_sample):
    # Convert NumPy array to RDKit BitVector
    gan_rdkit_fp = DataStructs.ExplicitBitVect(len(gan_fp))
    for k, bit in enumerate(gan_fp):
        if bit == 1:
            gan_rdkit_fp.SetBit(k)
```

```
    similarities = []
    for j, real_fp in enumerate(fingerprints_array):
        real_rdkit_fp = DataStructs.ExplicitBitVect(len(real_fp))
        for k, bit in enumerate(real_fp):
            if bit == 1:
                real_rdkit_fp.SetBit(k)
        sim = DataStructs.TanimotoSimilarity(gan_rdkit_fp, real_rdkit_fp)
        similarities.append((j, sim))

    similarities.sort(key=lambda x: x[1], reverse=True)
    print(f"\n GAN Sample {i+1}     Most similar ALS drug index: {similarities[0][0]} (similarity: {
        similarities[0][1]:.3f})")
    print("Drug Name:", als_drugs_valid.iloc[similarities[0][0]]["Drug Name"])

# Convert non-ALS SMILES to fingerprints
non_als_df['Fingerprint'] = non_als_df['SMILES'].apply(smiles_to_morgan_fp)
non_als_df = non_als_df.dropna(subset=['Fingerprint'])

# For each GAN fingerprint, find the top-3 most similar non-ALS drugs
results = []

for i, gan_fp_array in enumerate(fake_fps_binary[:5]):
    gan_fp = DataStructs.ExplicitBitVect(len(gan_fp_array))
    for k, bit in enumerate(gan_fp_array):
        if bit == 1:
            gan_fp.SetBit(k)

    sims = []
    for idx, row in non_als_df.iterrows():
        real_fp_array = row['Fingerprint']
        real_fp = DataStructs.ExplicitBitVect(len(real_fp_array))
        for k, bit in enumerate(real_fp_array):
            if bit == 1:
                real_fp.SetBit(k)
        sim = DataStructs.TanimotoSimilarity(gan_fp, real_fp)
        sims.append((row["Drug Name"], sim))

    sims.sort(key=lambda x: x[1], reverse=True)
    top_matches = sims[:3]

    for rank, (drug_name, sim_score) in enumerate(top_matches, 1):
        results.append({
            "GAN Sample": f"GAN {i+1}",
            "Rank": rank,
            "Closest Non-ALS Drug": drug_name,
            "Tanimoto Similarity": round(sim_score, 3)
        })

# Convert to DataFrame and preview
similar_non_als_df = pd.DataFrame(results)
print(similar_non_als_df)

# save to csv
similar_non_als_df.to_csv('/content/drive/My Drive/School Stuff/CPSC5440_ML/Project/GAN_NonALS_Similarity.
    csv', index=False)
```