January 2026

An Internship Report

Titled

Machine Learning-Enabled Drug Toxicity Prediction

by

Mr. Abhishek Kumar

Under the Supervision of

Dr. Tapan K Nayak,
Assistant Professor,
Kusuma School of Biological Sciences, IIT Delhi

Indian Institute of Technology, Delhi

TABLE OF CONTENTS

Chapter 1

Introduction

Drug development is a time-taking and expensive affair. The bench-to-market transition of a drug candidate, therefore, is a high-risk, low-success rate process [1]. Many approaches have been developed to speed it up and reduce costs e.g., high throughput *in silico* or *in vitro* screening [2]. Machine learning then becomes a natural choice when it comes to improving these existing approaches due to availability of cheaper computational resources, high iteration speed and readily available tools for code generation and development.

Machine learning (ML) is particularly valuable for predicting molecular properties, including toxicity [3].. Publicly available, experimentally-obtained toxicity datasets can be used to discern molecular properties of potentially toxic and safe compounds by training ML models to predict the properties from molecular structure or its representation [4]. This may permit improved efficiency in the identification of lead compounds as well as the evaluation of their safety profiles [5].

Safety assessment is key to lowering late-stage failures. As traditional experimental methods are expensive and time-consuming, computational approaches may act as ‘molecular sieves’ to identify potential compounds safe for therapeutic use and also, discard the toxic ones. As previously stated, machine learning (ML) methods can predict molecular toxicity by learning the relationship between structure and toxicity from experimentally tested compounds. Molecular representations, including physicochemical descriptors and structural fingerprints, could capture important chemical information and provide effective inputs for ML models for prediction of their safety profile [6]. Existing computational models often struggle to accurately capture complex structure–toxicity relationships, particularly in the presence of imbalanced datasets and diverse chemical structures, limiting their effectiveness in practical toxicity screening [7].

A molecular fingerprint is a compact numerical or binary representation of a molecule's structure, designed to enable fast similarity searching, clustering, and machine-learning in cheminformatics and drug discovery. Instead of working directly with full chemical graphs, fingerprints encode the presence or absence of structural features (atoms, bonds, substructures, or environments) into a fixed-length vector [8]. Table 1 shows a summary of different kinds of molecular fingerprints, their descriptions and use cases.

A typical molecular fingerprint generation workflow begins with obtaining the chemical structure of a compound. Atom-level features, such as elemental type, valence, hydrogen count, and aromaticity, are first assigned to each atom to create initial identifiers. These identifiers are then iteratively updated by incorporating information from the local neighbourhood of each atom within a predefined radius, capturing the surrounding chemical environment. The resulting substructures are subsequently encoded using a hashing scheme and mapped to a fixed-length binary representation, where the presence or absence of specific features is indicated by 1s and 0s. The final bit vector thus represents the molecular fingerprint of the compound. Figure 1 illustrates the overall fingerprint generation workflow. Figure 1 shows the typical workflow.

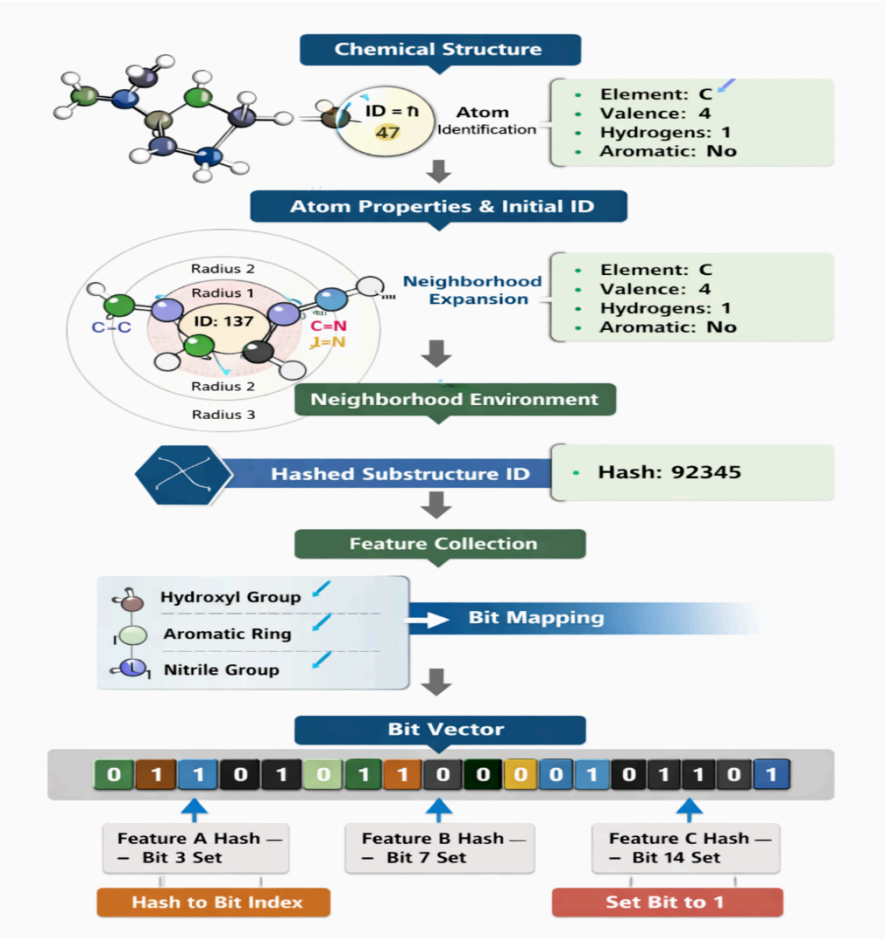


Figure 1 A typical molecular fingerprint generation process

Table 1 Common Types of Molecular Fingerprints *[9]*

Fingerprint Type	**Description**	**Use Case**	**Example**
Structural (Sub-structure)	Encodes the presence or absence of predefined chemical substructures using fixed keys.	Similarity search, basic QSAR studies	MACCS keys
Circular	Represents local atomic environments by expanding radially from each atom.	Toxicity prediction, drug discovery, ML models	Morgan (ECFP4)
Path-Based	Encodes linear atom paths up to a fixed bond length.	Structure comparison, similarity searching	Daylight fingerprints
Topological	Captures molecular connectivity and atom–bond relationships without 3D information.	Classical QSAR modeling	RDKit topological fingerprints

Pharmacophore	Encodes spatial arrangements of pharmacophoric features such as HBD, HBA, and aromatic rings.	Ligand-based virtual screening	Pharmacophore fingerprints
Physicochemical	Uses numerical molecular descriptors representing physicochemical properties.	Interpretable ML and baseline models	MW, LogP, TPSA
3D	Encodes three-dimensional molecular conformations and spatial relationships.	Structure-based drug design	3D pharmacophore finger-prints

Therefore, the goal of this work is to create a machine learning-based framework for predicting toxicity. This framework uses molecular representations to gather detailed substructural and chemical information. The features based on these fingerprints may encode important molecular patterns, which are then used for predicting toxicity. An ensemble-learning method using XGBoost serves as the baseline model for binary toxicity classification, and we are currently tuning hyperparameters to boost predictive performance.

Chapter 2

ML-based toxicity predictive models are usually categorized into traditional ML and emerging deep learning (DL) algorithms. Traditional ML algorithms include Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), and eXtreme Gradient Boosting (XGB). Deep Neural Networks (DNN), Graph Neural Networks (GNN), Recurrent Neural Networks (RNN), Graph Attention Network (GAT) [10].

XGBoost is an optimized implementation of gradient-boosted decision trees (GBDT), which combines multiple weak learners (e.g., DT) to form a strong model, particularly suitable for handling complex nonlinear relationships. In our

study, XGBoost was selected as the primary model because it has been consistently reported as one of the top-performing models in molecular toxicity prediction. Its frequent use in prior toxicity-related studies provides methodological reliability and comparability [11].

2.1 Model Formulation

XGBoost models predictions using an additive ensemble of decision trees. For a given input sample x_i, the predicted output $\{\backslash hat\{y\}\}_i$is expressed as:

$$\hat{y}_i = \sum_{k=1}^{K} f_k\left(x_i\right)$$

where f_k denotes the k-th decision tree in the ensemble and K is the total number of trees. Each tree maps the input feature vector $x_i$to a leaf node and outputs a real-valued score.

The learning objective of XGBoost is defined as

$$L = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i\right) + \sum_{k=1}^{K} \Omega\left(f_k\right)$$

where $l(\cdot)$is a differentiable loss function that measures the discrepancy between the true label $y_i$and the predicted value $\{\backslash hat\{y\}\}_i$, and $\backslash Omega(\cdot)$is a regularization term that penalizes model complexity.

The regularization term for a tree $f$is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where $T$is the number of leaf nodes in the tree, $w_j$is the score assigned to the j -th leaf, γ controls the penalty for adding a new leaf, and λ is the L2 regularization parameter.

XGBoost constructs the model in an additive manner. At boosting iteration t, the prediction is updated as:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t\left(x_i\right)$$

where $f_t$represents the decision tree added at the t-th boosting iteration.

To optimize the objective function efficiently, XGBoost applies a second-order Taylor expansion of the loss function around the current prediction $\hat{y}_i^{(1)}$:

$$l\left(y_i, \hat{y_i^{(t)}}\right) \approx l\left(y_i, \hat{y_i^{(t-1)}}\right) + g_i f_t\left(x_i\right) + \frac{1}{2} h_i f_t\left(x_i\right)^2$$

where the first-order gradient and second-order Hessian are defined as:

$$g_i = \frac{\partial l\left(y_i, \hat{y_i}\right)}{\partial \hat{y_i}},$$

$$h_i = \frac{\partial^2 l\left(y_i, \hat{y_i}\right)}{\partial \hat{y_i^2}}$$

By ignoring constant terms, the objective function at iteration $t$can be written as:

$$L^{(t)} = \sum_i \left[g_i f_t\left(x_i\right) + \frac{1}{2} h_i f_t\left(x_i\right)^2 \right] + \Omega\left(f_t\right)$$

Each decision tree assigns an input sample $x_i$to a specific leaf node using a mapping function $q(\backslash cdot)$, such that:

$$f_t\left(x_i\right) = w_{q\left(x_i\right)}$$

where $q\left(x_i\right) \in 1, 2, \ldots, T$ denotes the index of the leaf node to which sample $x_i$is assigned, and $w_j$is the score associated with the j-th leaf.

Substituting the tree representation into the objective function yields:

$$L^{(t)} = \sum_{j=1}^{T} \left[\sum_{i \in I_j} \left(g_i w_j + \frac{1}{2} h_i w_j^2 \right) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

where $I_j = i | q\left(x_i\right) = j$is the set of samples assigned to leaf j.

For notational simplicity, the aggregated gradient and Hessian for leaf $j$are defined as:

$$G_j = \sum_{i \in I_j} g_i,$$

$$H_j = \sum_{i \in I_j} h_i$$

Using these aggregated statistics, the objective function becomes:

$$L^{(t)} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} \left(H_j + \lambda \right) w_j^2 \right] + \gamma T$$

To obtain the optimal leaf weight w_j, the derivative of the objective with respect to $w_j$is set to zero:

$$\frac{\partial L^{(t)}}{\partial w_j} = G_j + \left(H_j + \lambda \right) w_j = 0$$

Solving for $w_j$gives:

$$w_j^* = - \frac{G_j}{H_j + \lambda}$$

Substituting the optimal leaf weight back into the objective yields the contribution of leaf $j$to the loss:

$$L_j = - \frac{1}{2} \frac{G_j^2}{H_j + \lambda}$$

For a candidate split that divides a node into left (L) and right (R) child nodes, the gain is defined as:

$$\text{Gain} = \frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right) - \gamma$$

After constructing the tree, the prediction for each sample is updated as:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta \, w_{q(x_i)}$$

where η ∈ (0,1] is the learning rate controlling the contribution of the newly added tree.

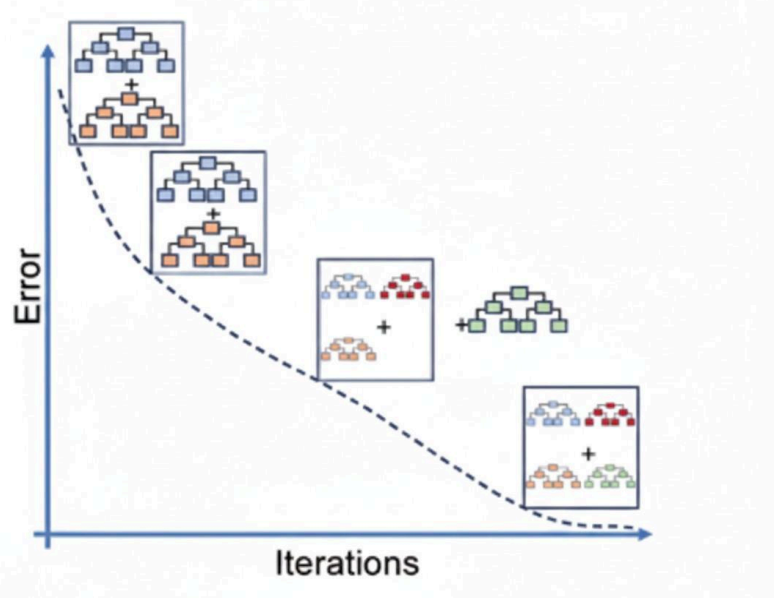


Figure 2 XGBoost operation - Reduction in error rate with every iteration[12].

Model Evaluation

2.2 General Information

The most used evaluation metrics for binary classification are accuracy, Recall , F1-score, and precision, which express the percentage of correctly classified instances in the set of all the instances, the truly positive instances, the truly negative instances, or the instances classified as positive, respectively. Sensitivity is commonly referred as recall. They have the formulas [13].

Let:

$y_i \in \{0, 1\}$ *denotes the true label of sample* i

$\hat{y}_i \in \{0, 1\}$ *denotes the predicted label*

$\hat{p}_i \in [0, 1]$ *denotes the predicted probability of the positive* $(toxic)$ *class*

N be the total number of samples

TP = True Positives, *TN* = True Negatives

FP = False Positives, *FN* = False Negatives then

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

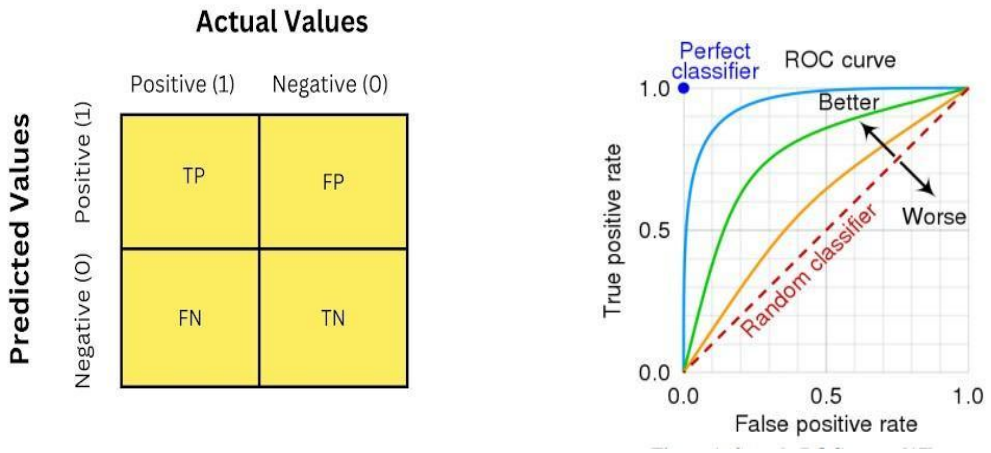
$$ROC - AUC = \int_0^1 TPR(f)\, d(FPR(f))$$

$$Recall = \frac{TP}{TP+FN}$$

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision+Recall}$$

.

The types of true/false positives/negatives are also represented in a confusion matrix format that explains away the specificity or sensitivity of a particular model. Sample ROC curves can also be plotted to visualize the improvement in model outputs with successive iterations, as shown in Figure 4.



(a) (b)

Figure 3 Important model evaluation parameters: a) Confusion matrix (https://h2o.ai/wiki/confusion-matrix/) b) Sample ROC curve [14]

Chapter 3

Model execution

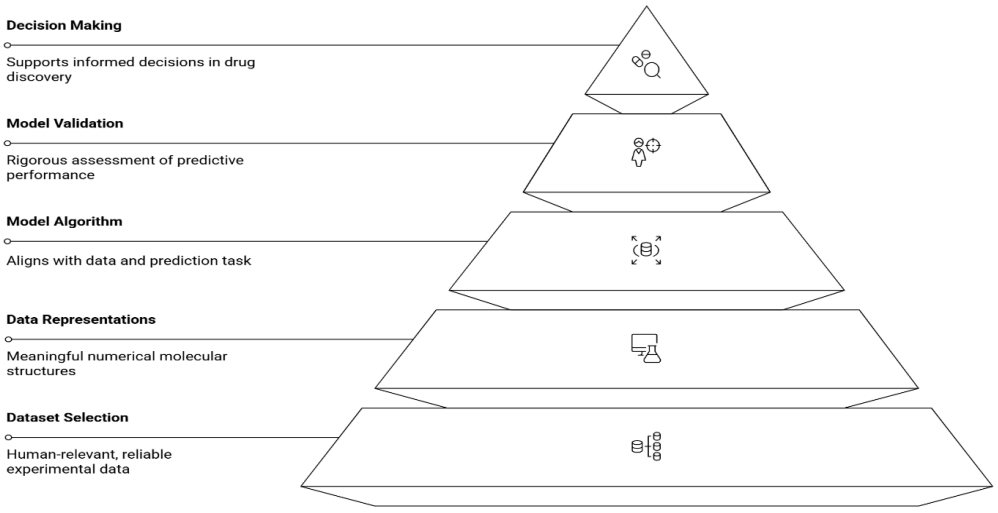


Figure 4 Hierarchical Workflow for Machine Learning–Driven Drug Discovery

3.1 System Setup

- All experiments were conducted on a Windows-based computational environment to ensure accessibility and ease of development.
- The implementation was carried out using Python version 3.10 within a dedicated Conda environment named rdkit_env, which provided isolation of dependencies and improved reproducibility of experiments.
- The Conda package manager was used for environment creation and library management, enabling consistent installation of scientific computing and cheminformatics packages.
- The RDKit library was employed for molecular structure processing, descriptor computation, and molecular fingerprint generation.
- To ensure compatibility with RDKit and related scientific libraries, the NumPy version was restricted to releases lower than 2.x.x, which provided stable performance and avoided known dependency conflicts.
- Core data processing and machine learning tasks were implemented using widely adopted Python libraries, including NumPy, pandas, scikit-learn, and XGBoost.
- Version control was Performed through GitHub (Private repository)

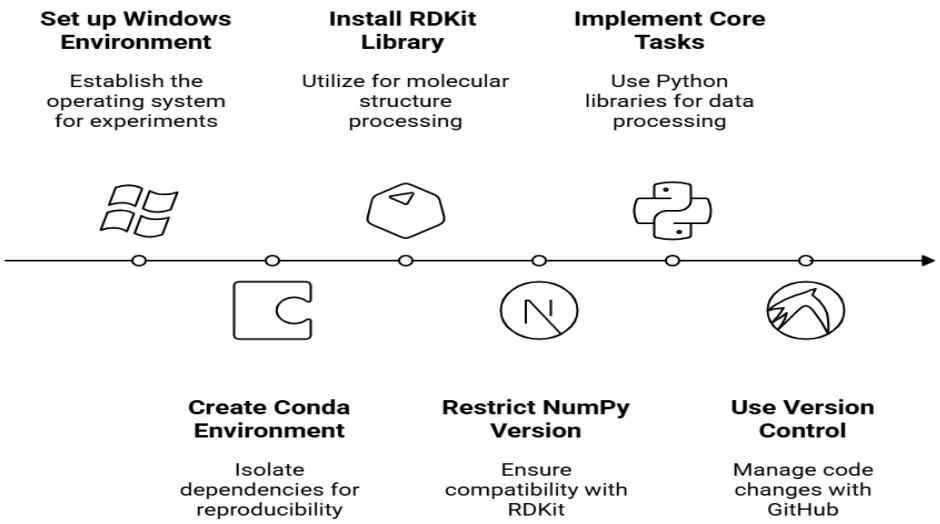


Figure 5 System Setup Workflow

3.2 Data Sources

- **Tox21 Dataset (Toxic Class):** Toxic compounds were obtained from the Tox21 dataset, a publicly available toxicity benchmark dataset generated through high-throughput screening assays.

  Dataset Size: 7831 rows × 14 columns

  Dataset link: https://tox21.gov/data-and-tools/

- **DrugBank FDA-Approved Drugs (Non-toxic Class):** Non-toxic compounds were sourced from the DrugBank database, consisting of FDA-approved drugs with established safety profiles.

  Dataset Size: 13166 rows × 17 columns

  Database link: https://go.drugbank.com/

- Molecular structures from both datasets were represented using SMILES notation and subsequently processed for feature extraction and fingerprint generation.

- The combined dataset enabled the construction of a binary classification framework distinguishing toxic and non-toxic compounds based on molecular structure.

Table 2 Relevant databases for data collection

Database Name	Class	Description	Link
GRAS Substances (SCOGS)	Non-toxic	FDA-recognized safe compounds for food and pharmaceutical use	Link
AYUSH Compounds Database	Non-toxic	Traditional Indian medicinal compounds with known safety profiles	Link
CLINTOX	Toxic	Clinical trial failure data due to toxicity	Link
ToxCast	Toxic	High-throughput in vitro toxicity screening assays (EPA)	Link
EPA DSSTox	Toxic	Curated chemical toxicity and exposure data	Link
Tox21	Toxic	Benchmark dataset for ML-based toxicity prediction	Link
SIDER	Toxic	Side-effect information extracted from drug labels	Link
Open TG-GATEs	Toxic	Toxicogenomics data for drug-induced toxicity	Link

Drug-Induced Liver injury (DILIrank)	Toxic	Human liver toxicity ranking of approved drugs	Link
FDA Adverse Event Reporting System (FDAERS)	Toxic	Post-marketing adverse drug reaction reports	Link
COSMOS Toxicology Database	Toxic	Repeated-dose and systemic toxicity data	Link
REACH (ECHA)	Toxic	Chemical hazard and safety data submitted under EU REACH	Link
TOXREFDB	Toxic	In vivo animal toxicity studies	Link
ChEMBL Toxicity Subsets	Mixed	Bioactivity and toxicity annotations from literature	Link
PubChem BioAssay	Mixed	Bioassay-based toxicity and activity screening	Link

3.3 Data Preprocessing

The toxic and non-toxic datasets were pre-processed independently due to differences in their source formats and data characteristics. The following preprocessing steps were applied:

1. **Data Collection and Curation:**

- **FDA-Approved Drug:** DrugBank entries belonging to the categories *Withdrawn*, *Experimental*, and *Illicit* were removed to ensure that only FDA-approved drug compounds were retained as non-toxic samples.

```
import pandas as pd

# keywords to remove
keywords = ['Withdrawn', 'Experimental',
'Illicit']

df_clean = df1[
                                    ~df1['Drug
Groups'].str.contains('|'.join(keywords),

case=False,

na=False)
]
```

- Irrelevant features were removed like
  BindingDB ID','ChemSpider ID','HET ID','ChEMBL ID','ChEBI ID

```
df1.drop(
    [ 'BindingDB ID', 'ChemSpider ID', 'HET
ID', 'ChEMBL ID', 'ChEBI ID', 'PubChem
Substance ID', 'PubChem Compound ID', 'KEGG
Drug ID', 'KEGG Compound ID', 'Formula',
'InChIKey', 'CAS Number', 'Name', 'DrugBank
ID'],
    axis=1,
    inplace=True
)
```

- Added a column named toxic and assigned 0 for each sample (non-toxic).
- Any rows containing missing values were excluded from the dataset.
- Left only SMILES and toxic columns in the dataset.

- **Tox21 Dataset:** Dropped all the columns except SMILES because integration of two datasets (common features) become easy.

```
df2.drop(
    [
        'NR-AR', 'NR-AR-LBD', 'NR-AhR', 'NR
-Aromatase', 'NR-ER', 'NR-ER-LBD',
'NR-PPAR-gamma', 'SR-ARE', 'SR-ATAD5',
'SR-HSE', 'SR-MMP', 'SR-p53', 'mol_id'
  ],
    axis=1,
    inplace=True
)
```

- Added a new column called toxic assigned 1 for each sample (toxic).

2. **Concatenation:** concatenated Both (FDA Approved and Tox21).This is the Final pre-processed manually but for model application we need to do more improvement.

3.4 Sanitization and Salt Removal

- Salt removal was performed using RDKit to eliminate counter ions and retain only the largest fragment of each molecule, ensuring consistent molecular representations.
- Molecular sanitization was subsequently applied to validate chemical correctness, including checks for valence consistency and aromaticity perception.

- These preprocessing steps reduced structural noise, prevented invalid molecular representations, and improved the reliability of downstream fingerprint generation and toxicity prediction models.

  Initial molecules: 14,284
  Dropped molecules: 1,003
  Remaining molecules: 13,281
  Percentage dropped: 7.02%
- Duplicate entries were removed.

```
from rdkit import Chem from rdkit import RDLogger import
numpy as np

RDLogger.DisableLog('rdApp.*')

def is_organic(mol): return any(atom.GetSymbol()=='C' for
atom        in                mol.GetAtoms())        def
sanitize_and_desalt_smiles(smiles):

if not isinstance(smiles, str): try:

return None

mol = Chem.MolFromSmiles(smiles, sanitize=False) if mol
is None: return None

#   Split   into   fragments   frags=Chem.GetMolFrags(mol,
asMols=True,  sanitizeFrags=False)

# Keep only organic fragments organic_frags = [f for f in
frags if is_organic(f)] if not organic_frags: return None

# Choose largest organic fragment mol=max(organic_frags,
key=lambda m:  m.GetNumAtoms())

#  Sanitize  gently  try:  Chem.SanitizeMol(mol)  except:
Chem.SanitizeMol(mol,
sanitizeOps=Chem.SANITIZE_FINDRADICALS                  |
```

3.5 Molecular Descriptor Calculation

- Cleaned and desalted SMILES strings were converted into molecular graph objects using the RDKit cheminformatics library to ensure chemical validity.

```
from rdkit import Chem
from rdkit.Chem import Descriptors, rdMolDescriptors
import pandas as pd
from typing import List, Optional
import warnings

def calculate_compound_properties(
    smiles_list: List[str],
    df_clean: pd.DataFrame,
    names: Optional[List[str]] = None,
    add_hydrogens: bool = True
) -> pd.DataFrame:
    """
    Calculate molecular properties from SMILES strings using RDKit
    and attach Toxic column from df_clean.
    """
    results = []
    invalid_smiles = []

    for idx, smiles in enumerate(smiles_list):
                name = names[idx] if names and idx < len(names) else
f"CPMD_{idx+1:04d}"

        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            invalid_smiles.append((name, smiles))
            continue

        try:
            if add_hydrogens:
                mol = Chem.AddHs(mol)

            props = {
                "Compound": name,
                "MolecularWeight": Descriptors.MolWt(mol),
                "LogP": Descriptors.MolLogP(mol),
                "HBD": Descriptors.NumHDonors(mol),
                "HBA": Descriptors.NumHAcceptors(mol),
                "TPSA": rdMolDescriptors.CalcTPSA(mol),
                "RotatableBonds": Descriptors.NumRotatableBonds(mol)
            }

            results.append(props)

        except Exception as e:
            warnings.warn(f"Property calculation failed for {name}: {e}")

    df_props = pd.DataFrame(results)

    if "Toxic" in df_clean.columns:
        df_props["Toxic"] = df_clean["Toxic"].values[:len(df_props)]

    return df_props
```

Continue….

```
            properties = {
                "Name": name,
                "SMILES": smiles,
                "MW": round(Descriptors.MolWt(mol), 2),
                "LogP": round(Descriptors.MolLogP(mol), 2),
                "TPSA": round(Descriptors.TPSA(mol), 2),
                "RotBonds": rdMolDescriptors.CalcNumRotatableBonds(mol),
                "HBA": rdMolDescriptors.CalcNumLipinskiHBA(mol),
                "HBD": rdMolDescriptors.CalcNumLipinskiHBD(mol),
                "RingCount": rdMolDescriptors.CalcNumRings(mol),
                "HeavyAtoms": mol.GetNumHeavyAtoms(),
                "Aromatic": rdMolDescriptors.CalcNumAromaticRings(mol),
            }

            results.append(properties)

        except Exception as e:
            invalid_smiles.append((name, smiles, str(e)))
            warnings.warn(f"Error processing {name}: {e}")

    # Create properties dataframe
    prop_df = pd.DataFrame(results)

    df_unique = df_clean[['SMILES', 'Toxic']].drop_duplicates(subset='SMILES')

    prop_df = prop_df.merge(
        df_unique[['SMILES', 'Toxic']],
        on='SMILES',
        how='left',
        validate='one_to_one'
    )

    # Summary
    print(f"✓ Processed {len(prop_df)} valid compounds")
    if invalid_smiles:
        print(f"✗ Skipped {len(invalid_smiles)} invalid SMILES")

    return prop_df
```

- Molecules that failed SMILES parsing or produced errors during descriptor computation were identified and excluded from further analysis to maintain data integrity.
- Explicit hydrogen atoms were added to each molecule prior to descriptor calculation to improve the accuracy of physicochemical property estimation.
- A set of physicochemical molecular descriptors was calculated. These included molecular weight (MW), octanol-water partition coefficient (LogP), topological polar surface area (TPSA), number of rotatable bonds,

hydrogen bond acceptors (HBA), hydrogen bond donors (HBD), ring count, heavy atom count, and number of aromatic rings [8].

- Duplicate SMILES entries were removed prior to label integration to ensure a one-to-one correspondence between molecular structures and toxicity annotations.

3.6 Molecular Fingerprint Generation

```
import pandas as pd
import numpy as np
from rdkit import Chem
from rdkit.Chem import AllChem

prop_df = prop_df[prop_df['SMILES'].notna()]
prop_df['SMILES'] = prop_df['SMILES'].astype(str)

def smiles_to_morgan(smiles, radius=2, n_bits=2048):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    fp = AllChem.GetMorganFingerprintAsBitVect(
        mol,
        radius=radius,
        nBits=n_bits
    )
    return np.array(fp)

# Convert SMILES to fingerprints
prop_df['fingerprint'] = prop_df['SMILES'].apply(smiles_to_morgan)

# Remove invalid SMILES (if any)
prop_df = prop_df.dropna(subset=['fingerprint'])

prop_df.drop(['SMILES', 'Name'], axis=1, inplace=True)
```

- SMILES were used as the initial molecular representation and were cleaned by removing missing or invalid entries.
- Each valid SMILES string was converted into an RDKit molecular object to enable structural analysis.
- Morgan fingerprints (ECFP4) were generated with a radius of 2 and a fixed length of 2048 bits to encode local chemical environments.
- Molecules that failed fingerprint generation were excluded to ensure chemical validity.

- The resulting fingerprints were converted into numerical vectors and non-numerical identifiers were removed, producing a machine-learning–ready dataset.

Table 3 *Molecular descriptors and Morgan fingerprint representations of the curated dataset (13,183 compounds). The fingerprint column represents a 2048-bit binary vector*

Index	MW (Da)	LogP	TPSA (Å²)	RotBonds	HBA	HBD	RingCount	HeavyAtoms	Aromatic	Toxic	Fingerprint
0	2180.32	-8.12	901.57	72	57	31	6	155	3	0	[0,1,0,0,0,0..]
1	1269.43	-3.11	495.89	35	32	20	6	91	4	0	[0,1,0,0,0,0..]
2	1811.25	4.87	519.89	68	35	20	8	131	8	0	[0,1,0,0,0,0..]
3	1069.24	-4.13	435.41	20	26	18	4	74	2	0	[0,1,0,0,0,0..]
4	1431.06	-0.51	495.67	44	31	20	6	102	5	0	[0,1,0,0,0,0…]
…	…	…	…	…	…	…	…	…	…	…	…
13178	440.46	4.03	118.81	8	9	2	5	33	5	1	[0,0,0,0,0,0..]
13179	314.47	4.72	34.14	4	2	0	4	23	0	1	[0,0,0,0,0,0..]
13180	288.43	3.88	37.30	3	2	1	4	21	0	1	[0,0,0,0,0,0..]
13181	272.39	3.61	40.46	3	2	2	4	20	1	1	[0,0,0,0,0,0..]
13182	339.39	3.09	40.16	4	5	0	5	25	2	1	[0,0,0,0,0, 0,…]

3.7 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is the step in understanding a dataset before building any model. Its goal is to summarize, visualize, and inspect data to discover patterns, problems, and insights.

3.7.1 Description of numerical data

.

Table 4 Description of numerical data

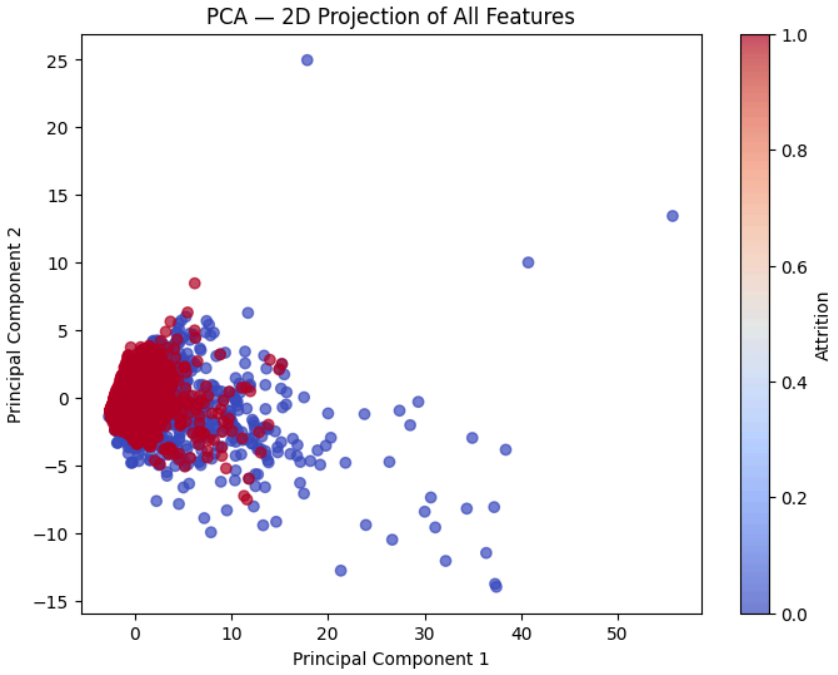
Feature	Count	Mean	Std Dev	Min	25%	50%	75%	Max
MW	13183	351.43	264.68	12.01	199.04	306.70	424.50	6552.56
LogP	13183	2.46	2.84	35.16	1.19	2.59	3.96	57.75
TPSA	13183	82.14	100.54	0.00	35.53	63.32	98.89	2202.68
RotBonds	13183	8.21	9.14	0.00	4.00	6.00	10.00	237.00
HBA	13183	5.71	6.41	0.00	3.00	4.00	7.00	173.00
HBD	13183	2.11	3.80	0.00	0.00	1.00	3.00	82.00
RingCount	13183	2.43	2.09	0.00	1.00	2.00	4.00	52.00
HeavyAtoms	13183	24.03	18.17	1.00	13.00	21.00	30.00	417.00
Aromatic	13183	1.47	1.36	0.00	0.00	1.00	2.00	24.00

3.7.2 PCA (2D)

Reduces high-dimensional molecular features to two dimensions using PCA and visualizes how toxic and non-toxic samples are distributed in that reduced space.

- Significant overlap between toxic and non-toxic samples indicates that the classes are not linearly separable in the PCA-reduced space.
- Dense central clustering suggests that most compounds share similar global physicochemical properties.
- Presence of outliers reflects molecules with distinct structural or chemical characteristics.

Figure 6 PCA showing overlap between classes

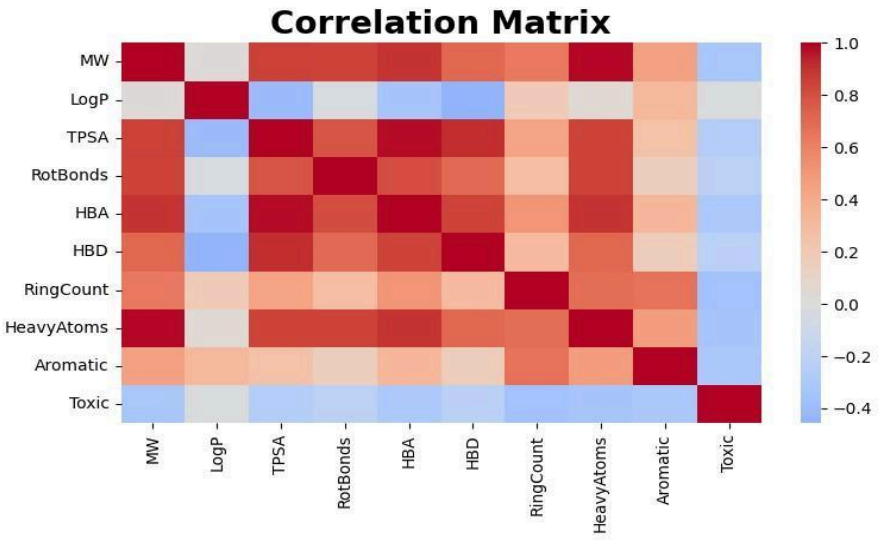



```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Separate features and target
X = df3.drop(columns=['Toxic'])
y = df3['Toxic']

# Standardize numeric features
X_scaled = StandardScaler().fit_transform(
X.select_dtypes(include=['float64', 'int64'])
)

# PCA to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Scatter plot of 2D projection
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='coolwarm', alpha=0.7)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA - 2D Projection of All Features')
plt.colorbar(label='Toxic')
plt.show()
```

- Non-linear modelling requirement is implied, motivating the use of advanced machine learning and deep learning approaches.

3.7.3 Correlation heatmap

Figure 7 Correlation Heatmap.



- Strong inter-feature correlation exists among molecular size descriptors such as MW, Heavy atoms, and TPSA.
- LogP provides complementary information due to its weaker correlation with other features.
- Toxicity shows low linear correlation with individual descriptors, indicating complex underlying relationships.
- Descriptor redundancy motivates the use of robust, non-linear machine learning models.

3.8 Model Development Code

- Imported Essential Python Libraries and Packages for Model Implementations and Evaluation.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import (
roc_auc_score,
accuracy_score,
recall_score,
classification_report,
confusion_matrix,
precision_score,
f1_score
)

from xgboost import XGBClassifier
from xgboost.callback import EarlyStopping
```

- Selected Descriptor columns and then named Xdesc
- Selected fingerprint Column which was in 1D array Converted into 2D array for Compatibility with Model.

```
# Descriptor columns
desc_cols = [
    "MW", "LogP", "TPSA", "RotBonds", "HBA", "HBD",
    "RingCount", "HeavyAtoms", "Aromatic"
]

# Prepare data
y = prop_df["Toxic"].astype(int).values
X_desc = prop_df[desc_cols].values
X_fp = np.vstack(prop_df["fingerprint"].values)


# Scale molecular descriptors only
scaler = RobustScaler()
X_desc_train = scaler.fit_transform(X_desc_train)
X_desc_test = scaler.transform(X_desc_test)

# Concatenate descriptors with fingerprints
X_train = np.hstack([X_desc_train, X_fp_train])
X_test = np.hstack([X_desc_test, X_fp_test])


X_desc_train,   X_desc_test,   X_fp_train,   X_fp_test,   y_train,   y_test   =
train_test_split(
    X_desc,
    X_fp,
    y,
    test_size=0.30,
    stratify=y,
    random_state=1001
)


# Class imbalance handling
neg, pos = np.bincount(y_train)
scale_pos_weight = neg / pos


# ==========================
# XGBoost Model
# ==========================
model = XGBClassifier(
    n_estimators=1500,
    max_depth=5,           # reduce depth
    min_child_weight=8,    # regularization
    gamma=0.3,             # split penalty
    learning_rate=0.03,
    subsample=0.7,
    colsample_bytree=0.7,
    reg_alpha=0.5,
    reg_lambda=2.0,
    scale_pos_weight=scale_pos_weight,
    objective="binary:logistic",
    eval_metric=["auc", "aucpr"],
    tree_method="hist",
    n_jobs=-1,
    random_state=42
)

# Train the model
model.fit(X_train, y_train)
```

•

Figure 8 XGBoost Hyperparameters and Their Roles.

Parameter	Value Used	Purpose and Explanation
n_estimators	1500	Number of boosting trees. A larger value allows the model to learn complex patterns gradually when combined with a small learning rate, improving generalization.
max_depth	5	Maximum depth of each tree. Controls model complexity by limiting how specific each tree can become, reducing overfitting [2].
min_child_weight	8	Minimum sum of Hessians required in a child node. Acts as regularization by preventing splits on small or noisy data subsets [2].
gamma	0.3	Minimum loss reduction required to make a split. Penalizes unnecessary splits, encouraging simpler and more robust trees [2].
learning_rate (η)	0.03	Step size shrinkage applied to each tree's contribution. Smaller values slow learning but improve generalization when using many trees.
subsample	0.7	Fraction of training samples used to build each tree. Introduces randomness, reduces variance, and helps prevent overfitting.
colsample_bytree	0.7	Fraction of features sampled for each tree. Improves robustness and reduces feature dominance, similar to Random Forests.
reg_alpha (α)	0.5	L1 regularization term on leaf weights. Encourages sparsity and reduces the impact of noisy features.
reg_lambda (λ)	2.0	L2 regularization term on leaf weights. Stabilizes weight updates and controls model complexity.
scale_pos_weight	$\frac{neg}{pos}$	Balances class imbalance by increasing the importance of the minority (toxic) class, improving recall for rare toxic samples.
objective	binary: logistic	Defines the learning task as binary classification with logistic loss, producing probabilistic outputs.
eval_metric	AUC, AUC-PR	Evaluation metrics chosen to assess ranking quality and performance on imbalanced datasets, particularly relevant for toxicity prediction.
tree_method	hist	Histogram-based tree construction for faster training and lower memory usage on large datasets.
n_jobs	-1	Utilizes all available CPU cores to accelerate
	x	ixtraining.
random_state	42	Ensures reproducibility of results by fixing the random seed.

3.8.1 Performance of My model

Table 5 Performance of the XGBoost model under different training–testing splits at threshold 0.21

Train. Data (%)	**Test. Data (%)**	**AUC–ROC**	**Accuracy**	**Recall**	**Precision**	**F1-score**
70	30	0.81	0.71	0.93	0.68	0.78
90	10	0.82	0.71	0.94	0.68	0.79
10	90	0.79	0.70	0.88	0.68	0.77
5	95	0.79	0.69	0.90	0.68	0.77

3.8.2 Most contributing Features for prediction.

- HeavyAtoms shows the highest importance, indicating that overall molecular size strongly influences toxicity prediction.
- RingCount, HBD, and MW also contribute significantly, suggesting that molecular structure and hydrogen bonding properties are key factors in toxic behavior.
- Several fingerprint features (e.g., FP_378, FP_786) appear among the top-ranked variables, highlighting the role of specific chemical substructures.
- The presence of both molecular descriptors and fingerprint features indicates that the model captures complementary global and local chemical information.
- Overall, the distributed importance across features suggests that toxicity arises from a combination of multiple molecular characteristics rather than a single dominant factor

Table 6 Top Features Ranked by XGBoost Gain Importance

Rank	Feature Index	Gain	Feature Name
1	7	23.839766	HeavyAtoms
2	387	15.115169	FP_378
3	6	12.152461	RingCount
4	5	11.632276	HBD
5	0	8.338769	MW
6	795	7.299445	FP_786
7	935	7.248703	FP_926
8	1625	6.801180	FP_1616
9	1123	6.762471	FP_1114
10	1937	6.675529	FP_1928
11	319	6.551643	FP_310
12	446	6.533344	FP_437

Chapter 4

Future Work

1. Although chemical structures are abstracted into fixed vectors by molecular fingerprints and descriptors, fine-grained structural information may be lost. Graph Neural Networks (GNNs), which work directly on molecular graphs and are better at capturing atom-bond relationships and message passing between atoms, may be investigated in future research [15].

2. Model performance can be enhanced by incorporating additional datasets such as **ToxCast, DSSTox, ClinTox, and GRAS compounds**, enabling better coverage of chemical space and reducing dataset bias [15].

3. To assess real-world applicability, future models should be validated on **external, unseen datasets** and evaluated under practical drug discovery scenarios, including early-stage screening pipelines [1].

References

[1] X. Qi, Y. Zhao, Z. Qi, S. Hou, and J. Chen, “Machine learning empowering drug discovery: Applications, opportunities and challenges,” *Molecules*, vol. 29, no. 4, p. 903, 2024.

[2] D. B. Kitchen, H. Decornez, J. R. Furr, and J. Bajorath, “Docking and scoring in virtual screening for drug discovery: methods and applications,” *Nat. Rev. Drug Discov.*, vol. 3, no. 11, pp. 935–949, 2004.

[3] L. Tonoyan and A. G. Siraki, “Machine learning in toxicological sciences: opportunities for assessing drug toxicity,” *Front. Drug Discov.*, vol. 4, p. 1336025, 2024.

[4] D. B. Catacutan, J. Alexander, A. Arnold, and J. M. Stokes, “Machine learning in preclinical drug discovery,” *Nat. Chem. Biol.*, vol. 20, no. 8, pp. 960–973, 2024.

[5] U. Gupta, A. Pranav, A. Kohli, S. Ghosh, and D. Singh, “The Contribution of Artificial Intelligence to Drug Discovery: Current Progress and Prospects for the Future,” in *Microbial Data Intelligence and Computational Techniques for Sustainable Computing*, vol. 47, A. Khamparia, B. Pandey, D. K. Pandey, and D. Gupta, Eds., in Microorganisms for Sustainability, vol. 47. , Singapore: Springer Nature Singapore, 2024, pp. 1–23. doi: 10.1007/978-981-99-9621-6_1.

[6] S. K. Niazi, “Artificial Intelligence in Small-Molecule Drug Discovery: A Critical Review of Methods, Applications, and Real-World Outcomes,” *Pharmaceuticals*, vol. 18, no. 9, p. 1271, 2025.

[7] S. Seal *et al.*, “Machine Learning for Toxicity Prediction Using Chemical Structures: Pillars for Success in the Real World,” *Chem. Res. Toxicol.*, vol. 38, no. 5, pp. 759–807, May 2025, doi: 10.1021/acs.chemrestox.5c00033.

[8] T. Yukawa and R. Naven, “Utility of Physicochemical Properties for the Prediction of Toxicological Outcomes: Takeda Perspective,” *ACS Med. Chem. Lett.*, vol. 11, no. 2, pp. 203–209, Feb. 2020, doi: 10.1021/acsmedchemlett.9b00536.

[9] J. Vamathevan *et al.*, “Applications of machine learning in drug discovery and development,” *Nat. Rev. Drug Discov.*, vol. 18, no. 6, pp. 463–477, 2019.

[10] C. Bai, L. Wu, R. Li, Y. Cao, S. He, and X. Bo, “Machine Learning-Enabled Drug-Induced Toxicity Prediction,” *Adv. Sci.*, vol. 12, no. 16, p. 2413405, Apr. 2025, doi: 10.1002/advs.202413405.

[11] W. Guo *et al.*, “Review of machine learning and deep learning models for toxicity prediction,” *Exp. Biol. Med.*, p. 15353702231209421, Dec. 2023, doi: 10.1177/15353702231209421.

[12] C. Bai, L. Wu, R. Li, Y. Cao, S. He, and X. Bo, “Machine Learning-Enabled Drug-Induced Toxicity Prediction,” *Adv. Sci.*, vol. 12, no. 16, p. 2413405, Apr. 2025, doi: 10.1002/advs.202413405.

[13] C. Miller, T. Portlock, D. M. Nyaga, and J. M. O’Sullivan, “A review of model evaluation metrics for machine learning in genetics and genomics,” *Front. Bioinforma.*, vol. 4, p. 1457619, 2024.

[14] S. Sathyanarayanan and B. R. Tantri, “Confusion matrix-based performance evaluation metrics,” *Afr. J. Biomed. Res.*, vol. 27, no. 4S, pp. 4023–4031, 2024.

[15] S. Seal *et al.*, “Machine Learning for Toxicity Prediction Using Chemical Structures: Pillars for Success in the Real World,” *Chem. Res. Toxicol.*, vol. 38, no. 5, pp. 759–807, May 2025, doi: 10.1021/acs.chemrestox.5c00033.