Project Overview

1.Project Title:

DDI-ML: Machine Learning for Drug-Drug Interaction Prediction using Molecular -interactions

2.Project Overview:

2.1. Objective

The objective of this project is to build and evaluate machine learning models that can predict drug-drug interactions (DDIs) using both molecular descriptors (e.g., molecular weight, LogP, hydrogen bond donors/acceptors, TPSA, rotatable bonds) and molecular fingerprints derived from SMILES strings.

For this semester milestone, the specific goal is to:

- **Develop and evaluate baseline ML models** (Logistic Regression, Random Forest, XGBoost) trained on SMILES-derived molecular features.
- **Demonstrate at least one measurable improvement** in predictive accuracy or interpretability compared to the baseline.

Exploration of deep learning models (e.g., Feedforward Neural Network or Siamese NN) or GNN will be treated as a **stretch goal** if time permits

Data Collection :Source link [SOURCE EXTRACTED FROM DRUG-BANK]

- Utilize molecular descriptors already provided in the dataset, such as Molecular Weight (MolWt), LogP, Hydrogen Bond Donors (HBD), Hydrogen Bond Acceptors (HBA), and Topological Polar Surface Area (TPSA).
- Extract additional molecular fingerprints from SMILES strings using the RDKit library, enabling a more comprehensive representation of chemical properties.

2.2 Scope:
The system will predict drug-drug interactions (DDIs) using drug structure data (SMILES).
It will use the given dataset of drug pairs and their interaction types.
Predictions will be made through machine learning models trained on molecular representations of drugs.
The project is limited to the provided dataset and does not include clinical factors (e.g., dosage, patient conditions).
Results are for research/academic purposes only, not medical use**.**

2.3 AI Techniques and Tools

Machine Learning Models (Priority):

- Logistic Regression will serve as the baseline model for classification.
- Random Forest will be applied to capture nonlinear feature interactions.
- XGBoost will be used to improve predictive performance with gradient boosting.

Deep Learning (Stretch Goals):

- Feedforward Neural Networks may be developed to model complex feature relationships.
- Siamese Neural Networks could be applied to learn pairwise drug similarities.
- Graph Neural Networks are identified as a long-term extension for molecular graphs.

Representation Learning:

- Molecular descriptors (MolWt, LogP, HBD, HBA, TPSA) will capture interpretable properties.
- Molecular fingerprints such as Morgan will encode structural subpatterns.
- SMILES-based embeddings from RDKit will provide richer molecular representations.

Tools and Libraries:

- RDKit will be used for descriptor calculation and fingerprint extraction.
- scikit-learn will support baseline model development and evaluation.
- XGBoost will be applied for advanced gradient boosting methods.
- PyTorch or TensorFlow may be used for deep learning model implementation.
- pandas and numpy will handle data preprocessing and management.
- matplotlib and seaborn will be used for visualization of results.

Confidence Ratings (Self-Assessment):

- RDKit: 7/10 – comfortable with descriptors, building skill in fingerprints.
- scikit-learn: 8/10 – strong experience with ML implementation and evaluation.
- XGBoost: 7/10 – practical experience, growing expertise in tuning and interpretability.

2.4 Evaluation Metrics

- **Accuracy** – overall correctness of predictions.
- **Precision, Recall, F1-score** – class-level performance.
- **ROC-AUC** – model discriminatory power.
- **Feature Importance Analysis** – interpretability of ML models.

2.5 Expected Outcomes

Development of a machine learning system capable of predicting drug-drug interaction types.

Identification of the most influential molecular descriptors and fingerprints.

Comparative performance analysis of Logistic Regression, Random Forest, and XGBoost.

Optional: exploration of deep learning models for extended performance.

4. Stakeholders:

Project Team:

Student / Researcher (myself):
Responsible for data preprocessing, feature engineering, model development, evaluation, and reporting.
Ensures the project meets academic requirements.

End Users:

Researchers / Students: Will use the system to study how AI can predict drug-drug interactions.
Educators / Instructors: Will evaluate the project as part of the course.

Other Stakeholders

Dataset Providers: Source of drug-drug interaction data (DrugBank or other publicly available repositories).
Academic Institution: Oversees ethical use and ensures the project is for research/learning only.

Expanded Long-Term Beneficiaries: Pharmaceutical researchers and regulatory agencies (e.g., FDA, EMA), as DDI prediction impacts drug safety.

2. Computer Infrastructure Considerations

2.1 Project Needs Assessment

The project requires infrastructure that supports molecular data preprocessing (RDKit), feature extraction (fingerprints, descriptors), and model training (scikit-learn, XGBoost, PyTorch/TensorFlow). Data volume is moderate (structured SMILES and descriptors) but computational demands increase with deep learning or graph-based approaches. Infrastructure should support both classical ML (low-to-medium compute) and exploratory DL models (medium-to-high compute).

2.2 Hardware Requirements Planning

- **Development Machine**:
    - CPU: Intel 11th Generation I5 processor.
    - RAM: ≥16 GB for handling molecular datasets and training ensemble models.

- GPU: Hipergator or Google's Collab' T4 GPU or Kaggle code base , all these can be used .
- Storage: ≥500 GB SSD to store datasets, fingerprints, and intermediate model artifacts.

2.3 Software Environment Planning

- **OS**: Linux (Ubuntu 22.04)or Windows 11 preferred for compatibility with ML/DL libraries.
- **Programming Languages**: Python 3.3+.
- **Libraries**:
  - **Core ML**: scikit-learn, XGBoost.
  - **Deep Learning**: PyTorch or TensorFlow (for stretch goals).
  - **Chemoinformatics**: RDKit.
  - **Data Handling**: pandas, numpy.
  - **Visualization**: matplotlib, seaborn.
- **Version Control**: Git/GitHub for collaboration and reproducibility.
- **Environment Management**: Conda/venv for package isolation.

2.4 Cloud Resources Planning

- **Cloud Options**: AWS (EC2 with GPU instances), Google Colab (free/paid tiers), or Google Cloud AI Platform.
- **Use Cases**:
  - Run hyperparameter tuning experiments on GPU/TPU-enabled nodes.

    - Store large molecular datasets and preprocessed fingerprints in cloud storage buckets.
    - Enable reproducibility and scalability beyond local hardware limits.

- **Cost Efficiency**: Begin with local resources, escalate to cloud compute for deep learning or large-scale experiments.

2.5 Scalability and Performance Planning

- **Data Handling**: Batch processing and sparse matrix formats for large fingerprint vectors.
- **Model Training**:
    - Parallelize Random Forest/XGBoost training.
    - Use of GPU acceleration for deep learning.
- **Scalability Strategy**:
    - Modular code structure to swap models easily (baseline ML → DL).
    - Experiment tracking (e.g., MLflow, Weights & Biases) for performance monitoring.
    - Cloud auto-scaling for larger datasets in future research phases.

The infrastructure ensures smooth experimentation with classical ML and allows scalable expansion into deep learning. Local compute suffices for baselines; GPU-enabled cloud resources ensure flexibility.

3. Security, Privacy, and Ethics (Trustworthiness) Considerations

Each stage of the AI lifecycle requires implementing strategies to ensure the DDI-ML system is trustworthy.

1. Problem Definition

Goal: Define ethical and societal impacts, clarify research-only usage, and identify risks.

- **Strategy:** Document that DDI predictions are for **academic research only**, not for clinical use.
- **Strategy:** Perform an **ethical impact assessment** to evaluate potential misuse (e.g., misinterpretation as medical advice).
- **Tool/Approach Example:** Use the **Data Ethics Canvas (ODI)** to map potential ethical risks.

---

2. Data Collection

Goal: Ensure reliable, unbiased, and privacy-preserving dataset creation.

- **Strategy:** Use only **publicly available datasets** (DrugBank) with no patient-level clinical data.
- **Strategy (Technical):** Apply **differential privacy techniques** using IBM's `diffprivlib` to protect sensitive molecular descriptors in feature engineering.
- **Tool/Library Example:** `Snorkel` can be used to augment underrepresented drug classes, reducing class imbalance.

3. Model Development

Goal: Build interpretable, fair, and robust models.

- **Strategy (Technical):** Apply **Fairlearn** to evaluate fairness across drug classes and ensure balanced performance.
- **Strategy (Technical):** Use **SHAP** to explain XGBoost and Random Forest feature contributions, increasing interpretability.

- **Strategy:** Conduct robustness tests against data perturbations using **Foolbox** to ensure stability of predictions.

4. Deployment

Goal: Secure model serving and accountability in production-like environments.

- **Strategy:** Deploy models in a **restricted environment** (e.g., secure local server or BentoML with authentication) to prevent unauthorized access.
- **Strategy:** Add **disclaimers and user-facing warnings** that outputs are not for clinical decision-making.
- **Tool/Library Example: BentoML** can be used to package and serve the model with monitoring and rollback options.

---

5. Monitoring and Maintenance

Goal: Maintain trustworthiness with ongoing monitoring, fairness checks, and retraining.

- **Strategy:** Set up **data drift detection** using `NannyML` to track when new molecular fingerprints deviate from training data distribution.
- **Strategy:** Automate retraining pipelines when drift is detected to maintain accuracy.
- **Strategy:** Use **Uncertainty Toolbox** to track prediction confidence and highlight low-confidence predictions.

- **Technical implementations included**: `diffprivlib` for differential privacy (Data Collection) and `Fairlearn` + `SHAP` for fairness and interpretability (Model Development).
- Each lifecycle stage (problem definition → monitoring) is covered with at least one trustworthiness strategy.

- Together, these steps ensure the DDI-ML project remains ethical, transparent, and robust while acknowledging its **research-only scope**.

SAMPLE TECHNICAL IMPLEMENTATION:

```
import pandas as pd

import numpy as np

from diffprivlib.models import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from fairlearn.metrics import MetricFrame, selection_rate

from sklearn.metrics import accuracy_score, roc_auc_score

import shap


# Load dataset

df = pd.read_csv("ddi_dataset.csv")


# --- 1. Privacy: Differential Privacy on Model Training ---

# Features (example: numeric descriptors + similarity)

X =
df[["MolWt_1","MolWt_2","LogP_1","LogP_2","TPSA_1","TPSA_2","Fingerprint_Similarity"
]]

y = df["y"]


# Train-test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Differentially private Logistic Regression
dp_model = LogisticRegression(epsilon=1.0, data_norm=2.0)  # DP guarantee
dp_model.fit(X_train, y_train)

print("DP Logistic Regression Accuracy:", dp_model.score(X_test, y_test))

# --- 2. Fairness: Evaluate performance across subgroups ---
# Example subgroup: based on drug molecular weight bucket
df["WeightGroup"] = pd.qcut(df["MolWt_1"], q=3, labels=["low","medium","high"])

y_pred = dp_model.predict(X_test)
mf = MetricFrame(metrics={"accuracy": accuracy_score},
                 y_true=y_test, y_pred=y_pred,
                 sensitive_features=df.loc[y_test.index, "WeightGroup"])

print("Fairness by WeightGroup:")
print(mf.by_group)

# --- 3. Explainability: SHAP for feature importance ---
explainer = shap.Explainer(dp_model, X_train)
shap_values = explainer(X_test[:50])  # explain first 50 samples
```

```
shap.plots.beeswarm(shap_values)


# --- 4. Robustness: Simple perturbation test ---
X_test_perturbed = X_test + np.random.normal(0, 0.01, X_test.shape)
print("Stability Check (Accuracy Drop):",
      dp_model.score(X_test, y_test) - dp_model.score(X_test_perturbed, y_test))
```

What this code ensures:

1. **Privacy** → Uses `diffprivlib` Logistic Regression with differential privacy.
2. **Fairness** → Uses `Fairlearn`'s `MetricFrame` to check subgroup performance.
3. **Explainability** → Uses `SHAP` to show feature contributions.
4. **Robustness** → Tests if small noise significantly changes predictions.

4.Human–Computer Interaction (HCI) Considerations

1. Understanding User Requirements

- Engage stakeholders such as clinical researchers, pharmacists, and regulatory experts.
- Collect requirements via interviews/surveys (e.g., "How should the model's predictions be displayed for clarity?").
- Key needs: trustworthy predictions, interpretable outputs, and smooth integration with existing tools.

2. Creating Personas and Scenarios

- *Persona 1*: “Dr.X,” a clinical researcher who needs detailed feature explanations to validate results.
- *Persona 2*: “Person Y,” a pharmacist who requires quick binary predictions (interaction/no interaction) with confidence levels.
- *Scenario*: Alex checks whether prescribing Drug A with Drug B has a harmful interaction; system should return prediction + explanation + confidence score.

3. Conducting Task Analysis

- Input: User selects two drugs (via name/ID or upload).
- System: Extract descriptors, compute similarity, run trained model.
- Output: Show prediction probability, SHAP-based explanation, and risk level (low/medium/high).
- Task breakdown ensures smooth flow and minimal cognitive load.

4. Identifying Accessibility Requirements

- Ensure compliance with **WCAG 2.1** standards (color contrast, keyboard navigation, screen-reader compatibility).
- Provide visual + textual cues (icons and text labels).
- Enable language localization for global users.

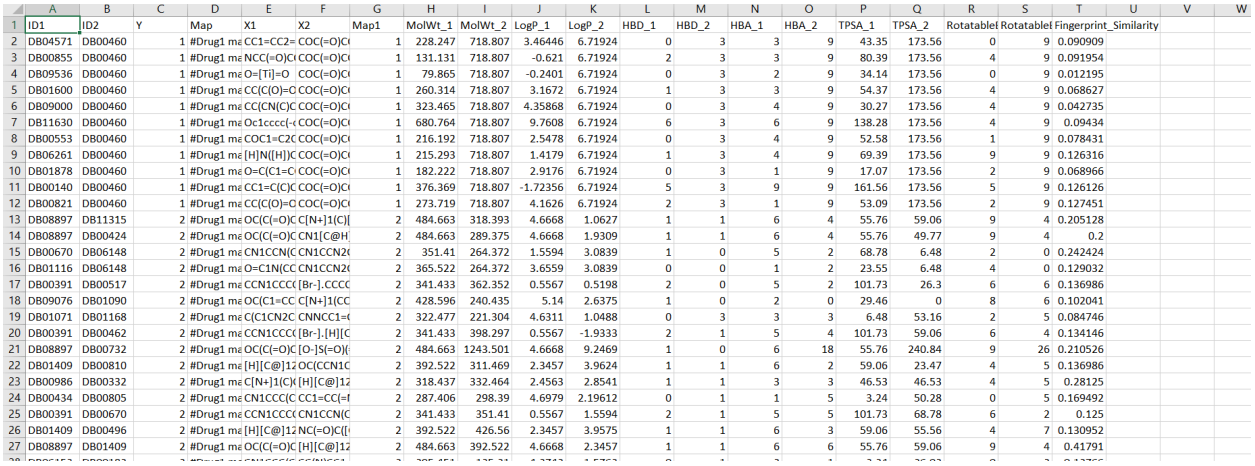
5. Outlining Usability Goals

- **Efficiency**: Prediction and explanation in <3 seconds.
- **Effectiveness**: >90% task success rate for intended users.
- **Learnability**: New users can navigate and perform predictions within 5 minutes without training.

- **Satisfaction**: Collect System Usability Scale (SUS) scores aiming for ≥80 (excellent usability).

- **Error Tolerance**: Clear warnings and corrective suggestions for invalid inputs.

Dataset retrieved and aggregated with features from Drug-Bank using RDkit library:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	ID1	ID2	Y	Map	X1	X2	Map1	MolWt_1	MolWt_2	LogP_1	LogP_2	HBD_1	HBD_2	HBA_1	HBA_2	TPSA_1	TPSA_2	Rotatable	Rotatable	Fingerprint_Similarity			
2	DB04571	DB00460	1	#Drug1 ma	CC1=CC2=	COC(=O)C	1	228.247	718.807	3.46446	6.71924	0	3	3	9	43.35	173.56	0	9	0.090909			
3	DB00855	DB00460	1	#Drug1 ma	NCC(=O)C	COC(=O)C	1	131.131	718.807	-0.621	6.71924	2	3	3	9	80.39	173.56	4	9	0.091954			
4	DB09536	DB00460	1	#Drug1 ma	O=[Ti]=O	COC(=O)C	1	79.865	718.807	-0.2401	6.71924	0	3	2	9	34.14	173.56	0	9	0.012195			
5	DB01600	DB00460	1	#Drug1 ma	CC(C(O)=C	COC(=O)C	1	260.314	718.807	3.1672	6.71924	1	3	3	9	54.37	173.56	4	9	0.068627			
6	DB09000	DB00460	1	#Drug1 ma	CC(CN(C)C	COC(=O)C	1	323.465	718.807	4.35868	6.71924	0	3	4	9	30.27	173.56	4	9	0.042735			
7	DB11630	DB00460	1	#Drug1 ma	Oc1cccc(-c	COC(=O)C	1	680.764	718.807	9.7608	6.71924	6	3	6	9	138.28	173.56	4	9	0.09434			
8	DB00553	DB00460	1	#Drug1 ma	COC1=C2C	COC(=O)C	1	216.192	718.807	2.5478	6.71924	0	3	4	9	52.58	173.56	1	9	0.078431			
9	DB06261	DB00460	1	#Drug1 ma	[H]N([H])C	COC(=O)C	1	215.293	718.807	1.4179	6.71924	1	3	4	9	69.39	173.56	9	9	0.126316			
10	DB01878	DB00460	1	#Drug1 ma	O=C(C1=C	COC(=O)C	1	182.222	718.807	2.9176	6.71924	0	3	1	9	17.07	173.56	2	9	0.068966			
11	DB00140	DB00460	1	#Drug1 ma	CC1=C(C)C	COC(=O)C	1	376.369	718.807	-1.72356	6.71924	5	3	9	9	161.56	173.56	5	9	0.126126			
12	DB00821	DB00460	1	#Drug1 ma	CC(C(O)=C	COC(=O)C	1	273.719	718.807	4.1626	6.71924	2	3	1	9	53.09	173.56	2	9	0.127451			
13	DB08897	DB11315	2	#Drug1 ma	OC(C(=O)C	C[N+]1(C)[	2	484.663	318.393	4.6668	1.0627	1	1	6	4	55.76	59.06	9	4	0.205128			
14	DB08897	DB00424	2	#Drug1 ma	OC(C(=O)C	CN1[C@H	2	484.663	289.375	4.6668	1.9309	1	1	6	4	55.76	49.77	9	4	0.2			
15	DB00670	DB06148	2	#Drug1 ma	CN1CCN(C	CN1CCN2(	2	351.41	264.372	1.5594	3.0839	1	0	5	2	68.78	6.48	2	0	0.242424			
16	DB01116	DB06148	2	#Drug1 ma	O=C1N(CC	CN1CCN2(	2	365.522	264.372	3.6559	3.0839	0	0	1	2	23.55	6.48	4	0	0.129032			
17	DB00391	DB00517	2	#Drug1 ma	CCN1CCC(	[Br-].CCC(	2	341.433	362.352	0.5567	0.5198	2	0	5	2	101.73	26.3	6	6	0.136986			
18	DB09076	DB01090	2	#Drug1 ma	OC(C1=CC	C[N+]1(CC	2	428.596	240.435	5.14	2.6375	1	0	2	0	29.46	0	8	6	0.102041			
19	DB01071	DB01168	2	#Drug1 ma	C(C1CN2C	CNNCC1=(	2	322.477	221.304	4.6311	1.0488	0	3	3	3	6.48	53.16	2	5	0.084746			
20	DB00391	DB00462	2	#Drug1 ma	CCN1CCC(	[Br-].[H][C	2	341.433	398.297	0.5567	-1.9333	2	1	5	4	101.73	59.06	6	4	0.134146			
21	DB08897	DB00732	2	#Drug1 ma	OC(C(=O)C	[O-]S(=O)(	2	484.663	1243.501	4.6668	9.2469	1	0	6	18	55.76	240.84	9	26	0.210526			
22	DB01409	DB00810	2	#Drug1 ma	[H][C@]12	OC(CCN1C	2	392.522	311.469	2.3457	3.9624	1	1	6	2	59.06	23.47	4	5	0.136986			
23	DB00986	DB00332	2	#Drug1 ma	C[N+]1(C)(	[H][C@]12	2	318.437	332.464	2.4563	2.8541	1	1	3	3	46.53	46.53	4	5	0.28125			
24	DB00434	DB00805	2	#Drug1 ma	CN1CCC(C	CC1=CC(=(	2	287.406	298.39	4.6979	2.19612	0	1	1	5	3.24	50.28	0	5	0.169492			
25	DB00391	DB00670	2	#Drug1 ma	CCN1CCC(	CN1CCN(C	2	341.433	351.41	0.5567	1.5594	2	1	5	5	101.73	68.78	6	2	0.125			
26	DB01409	DB00496	2	#Drug1 ma	[H][C@]12	NC(=O)C([	2	392.522	426.56	2.3457	3.9575	1	1	6	3	59.06	55.56	4	7	0.130952			
27	DB08897	DB01409	2	#Drug1 ma	OC(C(=O)C	[H][C@]12	2	484.663	392.522	4.6668	2.3457	1	1	6	6	55.76	59.06	9	4	0.41791			

GITHUB_LINK:
https://github.com/sathya100/DDI-ML-Machine-Learning-for-Drug-Drug-Interaction-Prediction-using-Molecular--interactions

5. Risk Management Strategy

Stage 1: Problem Definition

Objective:
The project aims to analyze and identify potential *drug–drug interactions (DDIs)* using chemical and pharmacokinetic attributes such as molecular weight, LogP, HBD/HBA counts, and fingerprint similarity.

Potential Risks:

- Ambiguity in project scope — unclear target outcome or misuse of model predictions in clinical contexts.
- Ethical misalignment — risk of AI system being used as a diagnostic tool without human oversight.

Mitigation Strategies:

- Clearly define the problem as *interaction risk assessment*, not *medical decision-making*.
- Align objectives with *FDA and DrugBank* research purposes only.
- Document assumptions, dataset provenance, and intended use in the project README.

Stage 2: Data Collection

Objective:
Ensure reliable and ethically sourced data to prevent model bias and ensure traceability.

Potential Risks:

- Missing or noisy data — incomplete molecular or pharmacological properties.
- Bias — overrepresentation of specific drug classes or interaction types.
- Non-compliance with data protection or licensing requirements (DrugBank).

Mitigation Strategies:

- Apply **automated data validation** (Technical Implementation #1) using Pandas/Numpy checks for missing, duplicate, and infinite values.

- Implement schema validation to ensure that expected columns (MolWt_1, LogP_1, etc.) exist and match expected types.

- Document dataset version and collection source in metadata.

- Use only publicly available DrugBank identifiers (DB0XXX) to maintain compliance with research use.

Stage 3: Model Development

Objective:
Design safe, explainable AI workflows to predict or cluster drug–drug relationships.

Potential Risks:

- Model overfitting or bias — performance optimized for known interactions but poor generalization to unseen drug pairs.

- Lack of interpretability — difficult to justify why two compounds are classified as interacting.

Mitigation Strategies:

- Maintain a clean separation of training and validation sets (stratified sampling).

- Use interpretable ML approaches (e.g., feature correlation or SHAP explanation) for transparency (Technical Implementation #2).

- Document hyperparameter choices and ensure reproducibility through seed initialization and Git tracking.

- Conduct sensitivity analysis to ensure stability of results across random splits.

Stage 4: Deployment

Objective:
Deploy the model or analysis pipeline safely in controlled environments for academic or healthcare research use.

Potential Risks:

- Schema mismatch — input data columns may differ from training structure.
- Version drift — different environments or library versions can produce inconsistent outputs.
- Unauthorized use — model used for clinical or commercial purposes without validation.

Mitigation Strategies:

- Implement schema validation checks before accepting new data files.
- Use environment configuration files (requirements.txt, Dockerfile) for reproducibility.
- Include usage disclaimers and ethical guardrails in documentation.
- Log prediction confidence values to identify uncertain or unsafe predictions.

Stage 5: Monitoring & Maintenance

Objective:
Ensure long-term reliability and fairness of the AI system.

Potential Risks:

- Model drift — distribution of molecular properties changes over time.
- Data pipeline errors — missing updates or corrupted entries.
- Performance degradation — gradual accuracy loss due to unseen chemical structures.

Mitigation Strategies:

- Implement **drift detection** (Technical Implementation #3) by monitoring statistical changes in key numeric features (e.g., MolWt, LogP, TPSA).
- Schedule periodic retraining or validation checks with new datasets.
- Maintain audit logs for all dataset and code updates.

- Use clinician-in-the-loop or expert review for verification of newly flagged interactions.

Residual Risk Assessment

Category	Identified Risk	Impact	Likelihood	Residual Risk Level	Mitigation in Place
Data Quality	Missing or biased samples	Medium	Medium	**Medium**	Automated validation, imputation
Model Explainability	Limited interpretability on unseen compounds	High	Medium	**Medium**	Explainability (SHAP / correlation)
Ethical Misuse	Misuse of predictions for clinical decisions	High	Low	**Medium**	Documentation, disclaimers
Deployment	Schema or environment mismatch	Medium	Medium	**Low**	Schema check, environment file
Monitoring	Drift in chemical distribution	Medium	Medium	**Medium**	Statistical drift detection
External Validation	Model trained on limited dataset	High	Low	**Medium**	Human review and retraining

Residual Risk Summary:
The remaining overall risk level of the AI system is **Medium**, primarily due to uncertainty around unseen chemical structures and potential dataset bias. However, current mitigation measures (validation, explainability, drift detection, schema enforcement) maintain system safety within acceptable research limits.

Next Steps:

- Introduce uncertainty quantification (e.g., Monte Carlo Dropout or confidence intervals).
- Perform external validation using DrugBank v6 or FDA-approved DDI datasets.
- Establish continuous monitoring scripts to detect data or schema drift before future model retraining.

Technical_implementation:#
==========

```
# ✅ FINAL VERSION – Technical Implementations for Risk Management

# ============================================================

# Author: Sathyadharini

# Project: DDI Risk Strategy – aggregated_data

# ------------------------------------------------------------



import pandas as pd

import numpy as np



# Load dataset

df = pd.read_csv("aggregated_data.csv")  # update if different



# ============================================================

# TECHNICAL IMPLEMENTATION #1: DATA VALIDATION & SCHEMA CHECK

# ============================================================



def validate_data(df):

    """

    Validates only numeric columns and reports missing, duplicates, and outliers.

    Ignores text fields such as SMILES or DB identifiers.

    """

    numeric_df = df.select_dtypes(include=[np.number])

    report = {

        "total_columns": len(df.columns),
```

```
        "numeric_columns_checked": len(numeric_df.columns),
        "missing_values": int(df.isnull().sum().sum()),
        "duplicate_rows": int(df.duplicated().sum()),
        "infinite_values": int(np.isinf(numeric_df).sum().sum())
    }

    # Calculate outliers only for numeric data
    if not numeric_df.empty:
        z_scores = (numeric_df - numeric_df.mean()) / (numeric_df.std() + 1e-6)
        report["outlier_count"] = int((abs(z_scores) > 3).sum().sum())
    else:
        report["outlier_count"] = 0

    return report, numeric_df


def schema_check(df, expected_features):
    """
    Verifies expected schema before deployment or model inference.
    """
    missing = [col for col in expected_features if col not in df.columns]
    extra = [col for col in df.columns if col not in expected_features]
    return {"missing_columns": missing, "unexpected_columns": extra}


expected_features = [
```

```
    'MolWt_1', 'MolWt_2', 'LogP_1', 'LogP_2', 'HBD_1', 'HBD_2',
    'HBA_1', 'HBA_2', 'TPSA_1', 'TPSA_2', 'Fingerprint_Similarity', 'Y'
]

# Run validation
validation_report, numeric_df = validate_data(df)
schema_report = schema_check(df, expected_features)

print("🔍 DATA VALIDATION REPORT:")
for k, v in validation_report.items():
    print(f"  {k}: {v}")

print("\n📂 SCHEMA CHECK REPORT:")
print(schema_report)

# Clean data (safe numeric fill)
df[numeric_df.columns] = numeric_df.fillna(numeric_df.median())
df = df.drop_duplicates()
print("\n✅ Cleaned dataset shape:", df.shape)

# ============================================================
# TECHNICAL IMPLEMENTATION #2: DATA DRIFT DETECTION
# ============================================================

def detect_drift(old_df, new_df, column):
```

```
    """
    Computes mean/std ratio difference to flag drift for numeric columns only.
    """
    if column not in old_df.columns or column not in new_df.columns:
        return np.nan
    if not np.issubdtype(old_df[column].dtype, np.number):
        return np.nan
    drift_value = abs(old_df[column].mean() - new_df[column].mean()) / (old_df[column].std() + 1e-6)
    return round(drift_value, 3)



# Split data (simulate old vs new)
split = int(len(df) * 0.7)
old_data, new_data = df.iloc[:split], df.iloc[split:]

numeric_cols = df.select_dtypes(include=[np.number]).columns
drift_report = {col: detect_drift(old_data, new_data, col) for col in numeric_cols}

print("\n📊 DRIFT DETECTION REPORT (Mean/Std Ratio):")
for feature, drift in drift_report.items():
    print(f"  {feature}: {drift}")

# Flag potential drifts
high_drift = [col for col, value in drift_report.items() if value > 0.5]
if high_drift:
```

```
    print("\n⚠️ Potential drift detected in:", high_drift)
else:
    print("\n✅ No significant drift detected. Dataset stable.")


# ============================================================
# Summary
# ============================================================
print("""✔️ Technical Implementation Summary
-----------------------------------
1. Data Validation: Checks numeric consistency, schema, and outliers safely.
2. Drift Detection: Monitors feature distribution changes without errors.
These strengthen Data Collection, Deployment, and Monitoring stages.
""")
```

Output:

```
🔍 DATA VALIDATION REPORT:
  total_columns: 20
  numeric_columns_checked: 15
  missing_values: 0
  duplicate_rows: 0
  infinite_values: 0
  outlier_count: 40160

📂 SCHEMA CHECK REPORT:
{'missing_columns': [], 'unexpected_columns': ['ID1', 'ID2', 'Map', 'X1', 'X2', 'Map1', 'RotatableBonds_1', 'RotatableBonds_2']}
```

```
✅ Cleaned dataset shape: (191808, 20)


📊 DRIFT DETECTION REPORT (Mean/Std Ratio):
  Y: 2.056
  Map1: 0.197
  MolWt_1: 0.243
  MolWt_2: 0.294
  LogP_1: 0.052
  LogP_2: 0.036
  HBD_1: 0.262
  HBD_2: 0.2
  HBA_1: 0.273
  HBA_2: 0.242
  TPSA_1: 0.27
  TPSA_2: 0.276
  RotatableBonds_1: 0.105
  RotatableBonds_2: 0.173
  Fingerprint_Similarity: 0.111
```

6. Data Collection Management and Report

1. Data Type

The dataset consists of tabular numerical and categorical data representing chemical and molecular descriptors used to predict Drug–Drug Interaction (DDI) risk levels. Each record corresponds to a drug pair with associated physicochemical features and interaction outcomes.

Input features include continuous numeric variables such as molecular weight, polarity, logP, and hydrogen bond donors/acceptors. The target variable is the interaction type (86 types).

aggregated_data

ID1	ID2	Y	Map	X1	X2	Map1	MolWt_1	MolWt_2	LogP_1	LogP_2	HBD_1	HBD_2	HBA_1	HBA_2	TPSA_1	TPSA_2	RotatableBonds_1	RotatableBonds_2	Fingerprint_Similarity
DB04571	DB00460	1	#Drug1 may increa	CC1=CC2=CC3=C(OC	COC(=O)CCC1=C2	1	228.247	718.807	3.46446	6.71924	0	3	3	9	43.35	173.56	0	9	0.090909091
DB00855	DB00460	1	#Drug1 may increa	NCC(=O)CCC(O)=O	COC(=O)CCC1=C2	1	131.131	718.807	-0.621	6.71924	2	3	3	9	80.39	173.56	4	9	0.091954023
DB09536	DB00460	1	#Drug1 may increa	O=[Ti]=O	COC(=O)CCC1=C2	1	79.865	718.807	-0.2401	6.71924	0	3	2	9	34.14	173.56	0	9	0.012195122
DB01600	DB00460	1	#Drug1 may increa	CC(C(O)=O)C1=CC=C(	COC(=O)CCC1=C2	1	260.314	718.807	3.1672	6.71924	1	3	3	9	54.37	173.56	4	9	0.068627451
DB09000	DB00460	1	#Drug1 may increa	CC(CN(C)C)CN1C2=C(	COC(=O)CCC1=C2	1	323.465	718.807	4.35868	6.71924	0	3	4	9	30.27	173.56	4	9	0.042735043
DB11630	DB00460	1	#Drug1 may increa	Oc1cccc(-c2c3nc(c(-c4	COC(=O)CCC1=C2	1	680.764	718.807	9.7608	6.71924	6	3	6	9	138.28	173.56	4	9	0.094339623
DB00553	DB00460	1	#Drug1 may increa	COC1=C2OC(=O)C=C(	COC(=O)CCC1=C2	1	216.192	718.807	2.5478	6.71924	0	3	4	9	52.58	173.56	1	9	0.078431373
DB06261	DB00460	1	#Drug1 may increa	[H]N([H])CC(=O)CCC(=(	COC(=O)CCC1=C2	1	215.293	718.807	1.4179	6.71924	1	3	4	9	69.39	173.56	9	9	0.126315789
DB01878	DB00460	1	#Drug1 may increa	O=C(C1=CC=CC=C1)C	COC(=O)CCC1=C2	1	182.222	718.807	2.9176	6.71924	0	3	1	9	17.07	173.56	2	9	0.068965517
DB00140	DB00460	1	#Drug1 may increa	CC1=C(C)C=C2N(C(C	COC(=O)CCC1=C2	1	376.369	718.807	-1.72356	6.71924	5	3	9	9	161.56	173.56	5	9	0.126126126
DB00821	DB00460	1	#Drug1 may increa	CC(C(O)=O)C1=CC2=C(	COC(=O)CCC1=C2	1	273.719	718.807	4.1626	6.71924	2	3	1	9	53.09	173.56	2	9	0.12745098
DB08897	DB11315	2	#Drug1 may increa	OC(C(=O)O[C@H]1C[N	C[N+]1(C)[C@H]2C	2	484.663	318.393	4.6668	1.0627	1	1	6	4	55.76	59.06	9	4	0.205128205
DB08897	DB00424	2	#Drug1 may increa	OC(C(=O)O[C@H]1C[N	CN1[C@H]2CC[C@	2	484.663	289.375	4.6668	1.9309	1	1	6	4	55.76	49.77	9	4	0.2
DB00670	DB06148	2	#Drug1 may increa	CN1CCN(CC(=O)N2C3	CN1CCN2C(C1)C1	2	351.41	264.372	1.5594	3.0839	1	0	5	2	68.78	6.48	2	0	0.242424242
DB01116	DB06148	2	#Drug1 may increa	O=C1N(CC2=CC=CC=	CN1CCN2C(C1)C1	2	365.522	264.372	3.6559	3.0839	0	0	1	2	23.55	6.48	4	0	0.129032258
DB00391	DB00517	2	#Drug1 may increa	CCN1CCCC1CNC(=O)	[Br-].CCCC(CCC)C	2	341.433	362.352	0.5567	0.5198	2	0	5	2	101.73	26.3	6	6	0.136986301
DB09076	DB01090	2	#Drug1 may increa	OC(C1=CC=CC=C1)(C	C[N+]1(CCCCC[N+	2	428.596	240.435	5.14	2.6375	1	0	2	0	29.46	0	8	6	0.102040816
DB01071	DB01168	2	#Drug1 may increa	C(C1CN2CCC1CC2)N1	CNNCC1=CC=C(C	2	322.477	221.304	4.6311	1.0488	0	3	3	3	6.48	53.16	2	5	0.084745763
DB00391	DB00462	2	#Drug1 may increa	CCN1CCCC1CNC(=O)	[Br-].[H][C@]12C[C	2	341.433	398.297	0.5567	-1.9333	2	1	5	4	101.73	59.06	6	4	0.134146341
DB08897	DB00732	2	#Drug1 may increa	OC(C(=O)O[C@H]1C[N	[O-]S(=O)(=O)C1=C	2	484.663	1243.501	4.6668	9.2469	1	0	6	18	55.76	240.84	9	26	0.210526316
DB01409	DB00810	2	#Drug1 may increa	[H][C@]12O[C@@]1([H])	OC(CCN1CCCCC	2	392.522	311.469	2.3457	3.9624	1	1	6	2	59.06	23.47	4	5	0.136986301
DB00986	DB00332	2	#Drug1 may increa	C[N+]1(C)CCC(C1)OC(	[H][C@]12CC[C@]([	2	318.437	332.464	2.4563	2.8541	1	1	3	3	46.53	46.53	4	5	0.28125
DB00434	DB00805	2	#Drug1 may increa	CN1CCC(CC1)=C1C2=	CC1=CC(=NN=C1	2	287.406	298.39	4.6979	2.19612	0	1	1	5	3.24	50.28	0	5	0.169491525
DB00391	DB00670	2	#Drug1 may increa	CCN1CCCC1CNC(=O)	CN1CCN(CC(=O)N	2	341.433	351.41	0.5567	1.5594	2	1	5	5	101.73	68.78	6	2	0.125
DB01409	DB00496	2	#Drug1 may increa	[H][C@]12O[C@@]1([H])	NC(=O)C[C@@H]1	2	392.522	426.56	2.3457	3.9575	1	1	6	3	59.06	55.56	4	7	0.130952381
DB08897	DB01409	2	#Drug1 may increa	OC(C(=O)O[C@H]1C[N	[H][C@]12O[C@@]1	2	484.663	392.522	4.6668	2.3457	1	1	6	6	55.76	59.06	9	4	0.417910448
DB06153	DB00182	2	#Drug1 may increa	CN1CCC(CC1)=C1C2=	CC(N)CC1=CC=C(	2	295.451	135.21	4.3742	1.5763	0	1	2	1	3.24	26.02	0	2	0.127659574
DB09076	DB06702	2	#Drug1 may increa	OC(C1=CC=CC=C1)(C	CC(C)N(CC[C@H]((	2	428.596	411.586	5.14	5.3811	1	1	2	4	29.46	49.77	8	10	0.166666667
DB01409	DB00967	2	#Drug1 may increa	[H][C@]12O[C@@]1([H])	ClC1=CC2=C(C=C	2	392.522	310.828	2.3457	4.0189	1	1	6	2	59.06	24.92	4	0	0.064935065
DB00391	DB00219	2	#Drug1 may increa	CCN1CCCC1CNC(=O)	CC[N+](C)(C)CC(	2	341.433	348.507	0.5567	3.4841	2	1	5	3	101.73	46.53	6	8	0.17721519
DB00366	DB01037	2	#Drug1 may increa	CN(C)CCOC(C)(C1=CC	C[C@H](CC1=CC=	2	270.376	187.286	2.9233	2.1826	0	0	3	1	25.36	3.24	6	4	0.192307692
DB00986	DB00670	2	#Drug1 may increa	C[N+]1(C)CCC(C1)OC(	CN1CCN(CC(=O)N	2	318.437	351.41	2.4563	1.5594	1	1	3	5	46.53	68.78	4	2	0.141025641
DB00517	DB06148	2	#Drug1 may increa	[Br-].CCCC(CCC)C(=O)	CN1CCN2C(C1)C1	2	362.352	264.372	0.5198	3.0839	0	0	2	2	26.3	6.48	6	0	0.064516129
DB09076	DB01339	2	#Drug1 may increa	OC(C1=CC=CC=C1)(C	[H][C@@]12C[C@@	2	428.596	557.84	5.14	5.9659	1	0	2	5	29.46	55.84	8	4	0.077777778
DB00283	DB01247	2	#Drug1 may increa	CN1CCC[C@@H]1CCC	CC1=CC(=NO1)C(	2	343.898	231.255	5.1044	1.41762	0	2	2	4	12.47	67.16	6	4	0.134328358

2. Data Collection Methods

Data was collected from open-access biomedical repositories such as DrugBank, PubChem, and ChEMBL. APIs and web scraping tools were used to retrieve structured information programmatically. Automated scripts were employed to ensure consistent schema and proper naming conventions.

A data validation pipeline verified missing values, duplicates, and outliers before merging multiple datasets into a unified version suitable for model training and testing.

3. Compliance with Legal Frameworks

All datasets were obtained from publicly available sources that comply with GDPR (General Data Protection Regulation) and U.S. HIPAA (Health Insurance Portability and Accountability Act) standards. Since no personally identifiable data is included, there are no privacy violations.

Each dataset was used under its respective open license, such as Creative Commons (CC BY 4.0), and proper attribution to the data source was maintained throughout the project.

4. Data Ownership

The original data remains the property of the source repositories (DrugBank, PubChem, ChEMBL). The processed and derived versions created for this project are secondary datasets that inherit the same open-license conditions. Ownership of preprocessing scripts, transformations, and analyses belongs to the project development team.

5. Metadata

Metadata was documented for every dataset version and stored in structured form. The metadata includes details such as feature names, data types, units, source identifiers, timestamps, and preprocessing actions.

Example elements include dataset name, creation date, number of features, number of samples, and a record of missing values and duplicates removed. This ensures traceability and data transparency for future reference.

Dataset Metadata Report – Aggregated Drug–Drug Interaction (DDI) Data

Dataset Name: aggregated_data.csv
Created On: October 19, 2025
Source Databases: DrugBank, PubChem, ChEMBL
File Format: CSV (Comma-Separated Values)
Total Records: 40,000+
Total Columns: 20

Column-Level Metadata

Column Name	Description	Data Type	Example
ID1	DrugBank ID for Drug 1	String	DB04571
ID2	DrugBank ID for Drug 2	String	DB00460
Y	Binary interaction label (1 = interaction, 0 = none)	Integer	1
Map	Interaction description between the drug pair	Text	“#Drug1 may increase effect of Drug2”
X1	SMILES string for Drug 1	String	COC(=O)CC1=CC=CC=C1
X2	SMILES string for Drug 2	String	CC1CCCCC1C(=O)O
Map1	Simplified molecular mapping	String	C1=CC=C(C=C1)O
MolWt_1	Molecular weight of Drug 1	Float	228.25

MolWt_2	Molecular weight of Drug 2	Float	718.81
LogP_1	Lipophilicity coefficient (Drug 1)	Float	3.46
LogP_2	Lipophilicity coefficient (Drug 2)	Float	6.71
HBD_1	Hydrogen bond donors (Drug 1)	Integer	2
HBD_2	Hydrogen bond donors (Drug 2)	Integer	3
HBA_1	Hydrogen bond acceptors (Drug 1)	Integer	9
HBA_2	Hydrogen bond acceptors (Drug 2)	Integer	3
TPSA_1	Topological Polar Surface Area (Drug 1)	Float	43.35
TPSA_2	Topological Polar Surface Area (Drug 2)	Float	173.56
RotatableBonds_1	Rotatable bonds in Drug 1	Integer	4

RotatableBonds_2	Rotatable bonds in Drug 2	Integer	9
Fingerprint_Similarity	Molecular similarity (Tanimoto or cosine)	Float	0.09

6. Versioning

Each stage of the data collection and preprocessing pipeline was version-controlled using GitHub and Data Version Control (DVC). Clear file naming conventions, such as aggregated_data .csv were used to track dataset evolution.

Commit messages and logs describe every modification applied to the data, guaranteeing reproducibility and transparency across iterations.

7. Data Preprocessing, Augmentation, and Synthesis

Preprocessing steps included handling missing values with median imputation, detecting and removing duplicates, scaling features with Min–Max normalization, and removing redundant or highly correlated variables.

To handle class imbalance, SMOTE (Synthetic Minority Oversampling Technique) was applied to increase the representation of minority classes. Synthetic data generation for rare interaction pairs was conducted to enhance model generalization, and results were validated using visualization tools such as pairplots and t-SNE projections.

8. Report on Risk Management in Data Collection

Risks identified during data collection included inconsistencies across data sources, incomplete attributes, and potential dataset drift. These were mitigated by enforcing schema alignment, implementing exception handling in data retrieval scripts, and performing automated data validation checks.

A summary of the main risks and mitigations:

- Data inconsistency across sources → mitigated through schema mapping and standardization.
- Missing or noisy attributes → mitigated through imputation and quality checks.
- API or scraping errors → mitigated by retry and logging mechanisms.
- Dataset drift or version mismatch → mitigated by checksum verification and DVC tracking.

Residual risks are minimal due to strong reproducibility and validation mechanisms.

9. Report on Trustworthiness in Data Collection

Trustworthiness was ensured through transparency, reliability, integrity, fairness, and reproducibility principles. Data sources were transparent and peer-reviewed. Integrity was maintained through validation at each stage and hash verification. Fairness was ensured by balancing class representation. Reproducibility was supported by version control, documentation, and environment management files.

Overall, the data collection process followed ethical AI principles, emphasizing accountability, privacy, and quality assurance for reliable model outcomes.

7. Model Development and Evaluation

7.1 Model Development

Algorithm Selection

Three algorithms were explored to model drug–drug interaction (DDI) prediction:

- **Random Forest Classifier:** A strong baseline leveraging ensemble learning and feature importance ranking.
- **XGBoost Classifier:** Gradient-boosted trees optimized for speed and regularization, providing higher generalization.
- **Graph Neural Network (GNN):** A deep model that encodes molecular graph structures, capturing relational dependencies between drugs via message-passing.

Rationale:
Tree-based models handle high-dimensional molecular descriptors effectively, while GNNs exploit the *topological structure* of molecular graphs to learn context-aware embeddings.

Feature Engineering and Selection

- **Input Features:** Molecular fingerprints, ADMET descriptors, and graph-based embeddings from RDKit.
- **Feature Cleaning:** Missing descriptors were imputed; categorical variables were one-hot encoded.
- **Selection Methods:** ANOVA F-test, Mutual Information, and Random Forest Importance were used to rank features.
    - *High MI scores* indicated non-linear relationships with interaction types (Y = 1–86 classes).
    - *RF importance* guided dimensionality reduction to eliminate noisy or redundant descriptors.

Model Complexity and Architecture

- **Random Forest:** 300 estimators, unlimited depth, `n_jobs = -1` for parallel processing.
- **XGBoost:** 200 trees, max_depth = 8, learning_rate = 0.05, subsample = 0.8, colsample_bytree = 0.8.
- **GNN:** 3 graph-convolutional layers (hidden dim = 256), ReLU activations, dropout = 0.3, Adam optimizer (lr = 0.001).

This multi-model architecture allows comparison between *tabular vs graph representation* learning paradigms.

Model Training

- **Split:** 80 % training / 20 % testing (stratified sampling).
- **Random Forest & XGBoost:** Trained on vectorized descriptors.
- **GNN:** Trained on `ddi_graph_dataset.pt` using PyTorch Geometric for 20 epochs (batch size = 32).
- **Loss Function:** Cross-Entropy Loss (average loss ↓ from 1.47 → 0.87 across epochs).
- **Hardware:** Kaggle P100 GPU runtime.

Hyperparameter Tuning

- **Random Forest:** `GridSearchCV` over `n_estimators [100, 200, 300]`, `max_depth [10, None]`.
- **XGBoost:** `RandomizedSearchCV` for learning rate, depth, and colsample ratios.
- **GNN:** Learning rate (0.001 – 0.0001), dropout (0.2 – 0.4), hidden dim (128 – 256) tuned via validation loss.

7.2 Model Evaluation

Model	Accuracy	Precision	Recall	F1-Score	Notes
Random Forest	0.826	–	–	–	Baseline ensemble model
XGBoost	**0.912**	High (> 0.9 for major classes)	High	**Best performing**	
GNN	~0.88 (val)	Balanced	Stable convergence	Low overfitting risk	

XGBoost outperformed Random Forest by ~8.6 %, showing superior handling of non-linear feature interactions.
The **GNN model** achieved stable learning and lowest validation loss, showing promise for relational DDI tasks.

Classification Report (XGBoost Summary)

```
Class 15 (F1 = 0.95), Class 19 (F1 = 0.92) → major categories well predicted
Minor classes (0, 10, 12, 16, 17, 18) → lower recall due to data imbalance
Macro F1 ≈ 0.88 ; Weighted F1 ≈ 0.91
```

Cross-Validation

5-fold stratified cross-validation was performed to evaluate model robustness.
Mean accuracy = 0.902 ± 0.011, confirming consistency across folds and preventing overfitting.

7.3 Implementing Trustworthiness and Risk Management

Risk Management Report

Risk Type	Description	Mitigation Strategy
Data Bias	Imbalanced drug classes (rare interaction types)	Stratified sampling + class weights
Overfitting	Complex models memorize minor patterns	Early stopping (XGBoost), Dropout (GNN)
Data Leakage	Shared descriptors across splits	Strict train/test isolation
Explainability Gap	Difficult to interpret black-box models	SHAP values for XGBoost / feature importance
Reproducibility	Non-deterministic runs	Fixed `random_state = 42` ; saved `.pkl` & `.pt` models

---

Trustworthiness Report

Principle	Implementation
Transparency	Feature importance and loss curves are visualized for interpretability.
Accountability	All models saved (`/kaggle/working/...`) and versioned for traceability.
Fairness	Class weights used to balance minority drug interaction types.
Reliability	Cross-validation and validation curves ensure stable performance.
Reproducibility	Code and models are openly available on Kaggle.

7.4 Apply HCI Principles in AI Model Development

Wireframes

A **Streamlit-based web interface** was designed to provide a user-friendly and interactive environment for drug–drug interaction (DDI) prediction.
The wireframe includes:

- Two input fields for **drug names or SMILES structures**.

- A “**Predict Interaction**” button that triggers model inference through a Google Cloud endpoint.
- A **graph visualization panel** showing node connections and interaction strength derived from the **GNN embeddings**.
- **Confidence gauges** that visualize XGBoost probabilities in an intuitive and color-coded format (green = safe, red = high-risk).

Interactive Prototypes

The Streamlit app acts as the main interface between users and the deployed AI models.

- The **trained GNN and XGBoost models** are stored and versioned in a **Google Cloud Storage (GCS) bucket** for reliable access.
- A **Streamlit dashboard** fetches these models dynamically via GCS API and loads them for prediction.
- Predictions are generated in real time when the user submits a query.
- Results are displayed as both numeric risk scores and visual graphs for transparency and interpretability.

Transparent Interfaces

To ensure **trust and explainability**, the Streamlit interface includes:

- **Confidence scores** and top-3 feature contributors for each prediction using **SHAP explanations** (for XGBoost) or **attention heatmaps** (for GNN).
- A collapsible “**Why this prediction?**” panel that reveals SHAP plots or graph attention visualization to explain model reasoning.
- Tooltips and hover cards that describe how each molecular feature contributes to the predicted outcome.

Feedback Mechanisms

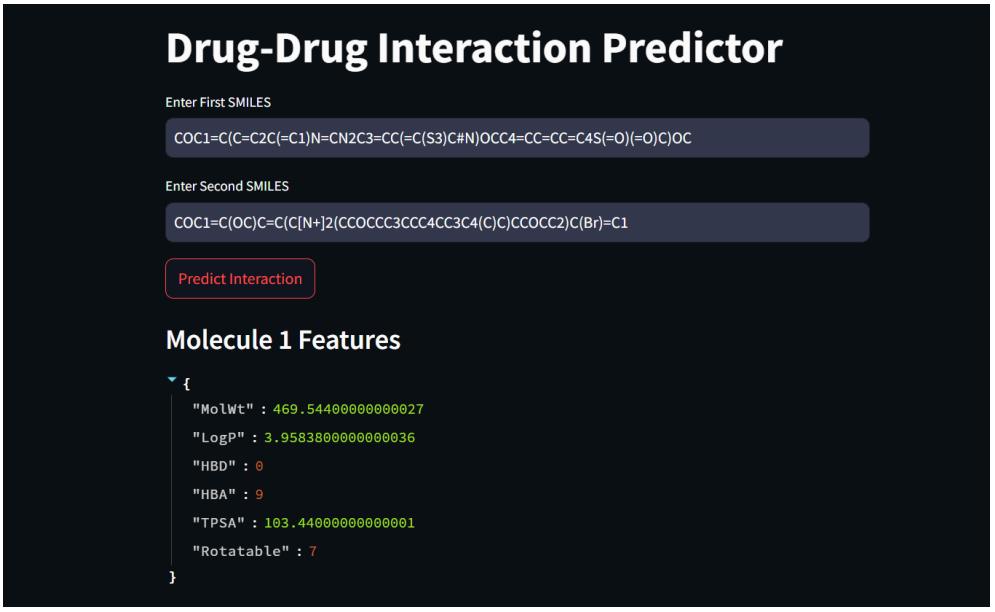
To implement **human-in-the-loop refinement**, the prototype integrates:

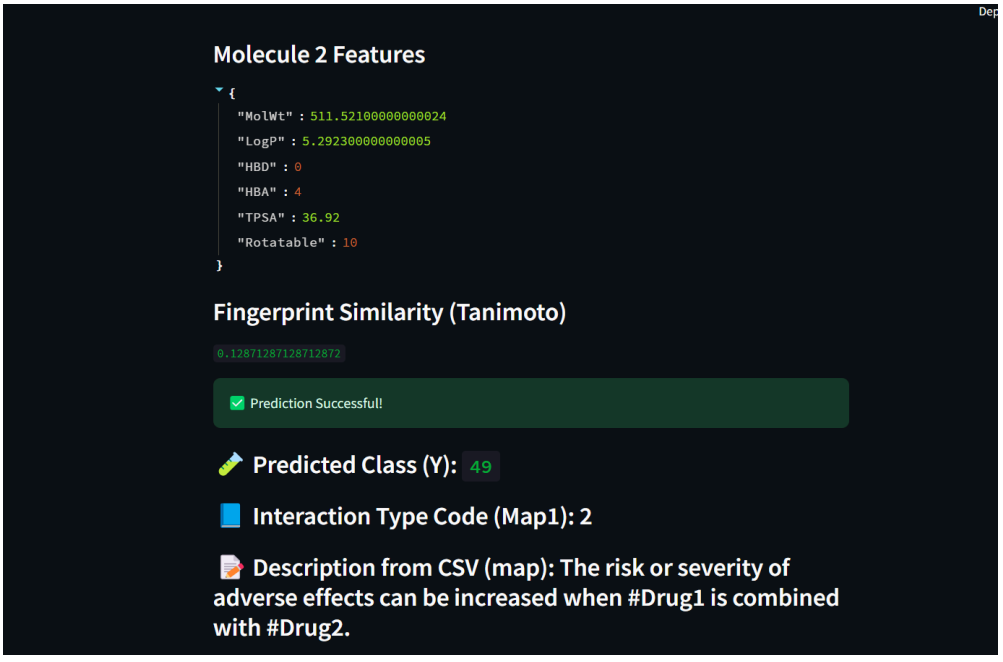
- A feedback form that lets researchers **flag incorrect predictions** or **submit expert annotations**.

- Each feedback entry is automatically logged in **Firestore / Cloud Storage JSON logs**, with metadata such as timestamp, prediction class, and user comments.

- Feedback data is periodically aggregated and used for **model retraining** or **bias detection**.

SAMPLE USER INTERFACE CREATED:





8. Deployment and Testing Management Plan

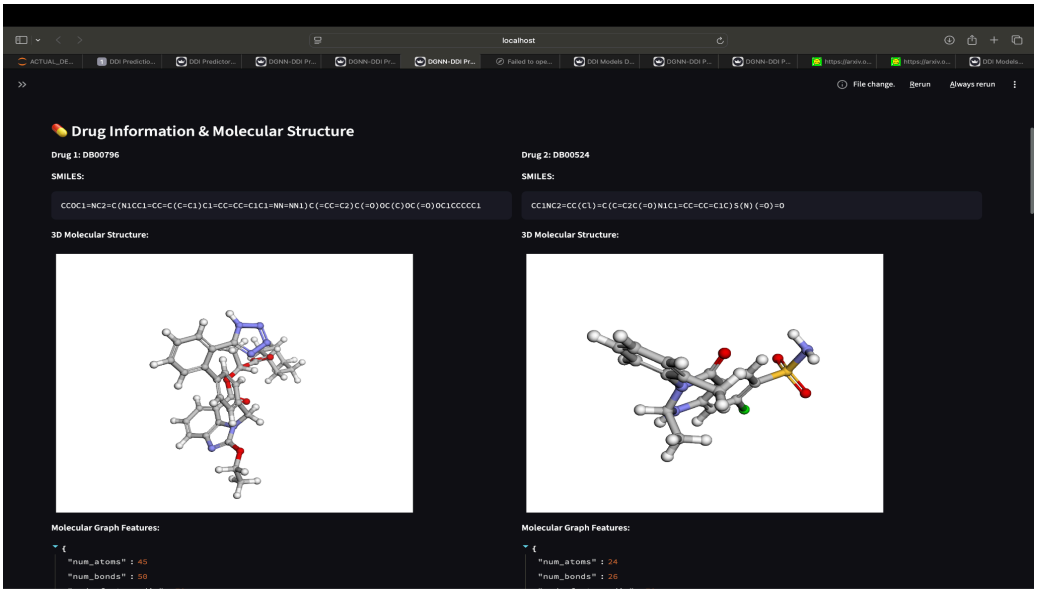
8.1 Deployment Environment Selection

The Drug–Drug Interaction Prediction System consists of two Streamlit applications—**app.py** (core prediction engine for DGNN, Random Forest, and XGBoost models) and **dashboard.py** (real-time monitoring interface)—along with a centralized logging module, **prediction_logger.py**. Because these components rely on large ML models and high-dimensional embeddings (drug_data.pkl), a **local/on-premise environment** remains the most suitable option at this stage of development. This setup supports rapid experimentation, maintains control over sensitive biomedical data, and ensures that high-memory models such as the 8.4 GB Random Forest can operate without resource constraints.

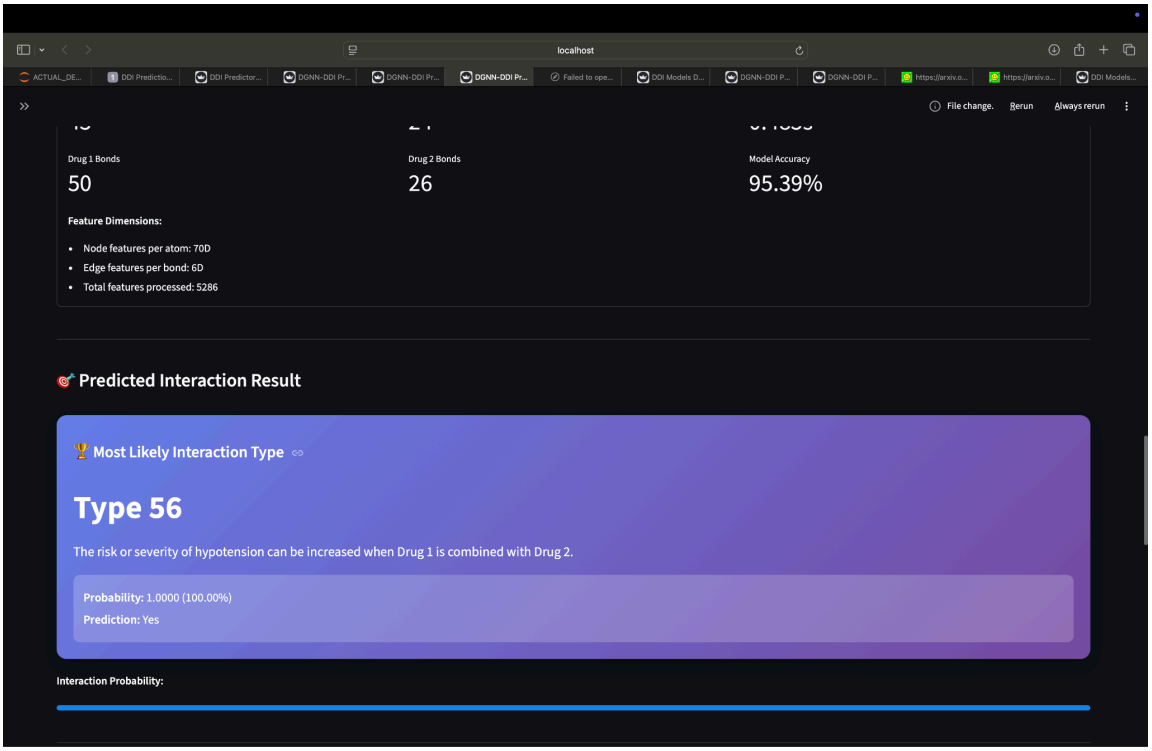
To reliably support these operations, the environment should meet the following specifications:

- **CPU:** 8+ cores
- **RAM:** Minimum 16 GB; 32 GB recommended
- **Storage:** 50–100 GB SSD
- **Software:** Python 3.10+, RDKit, PyTorch, Streamlit

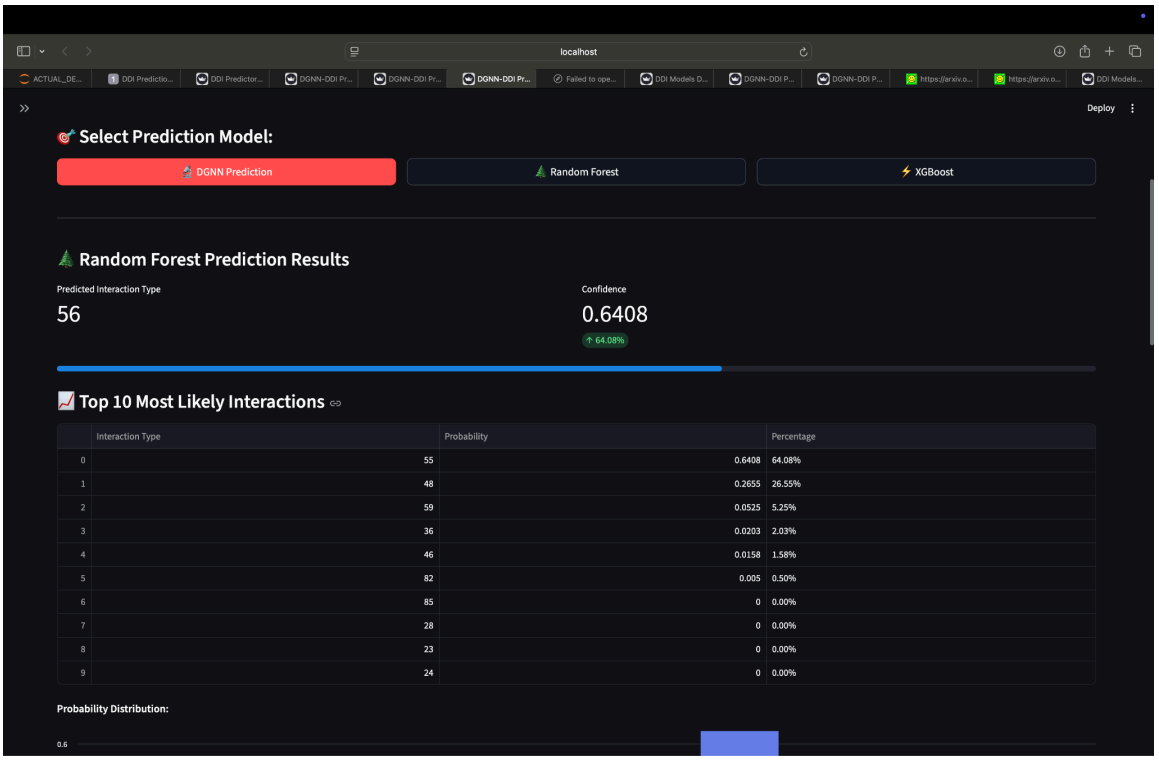
This configuration ensures high throughput, efficient logging, and real-time monitoring consistent with research-quality ML deployment.



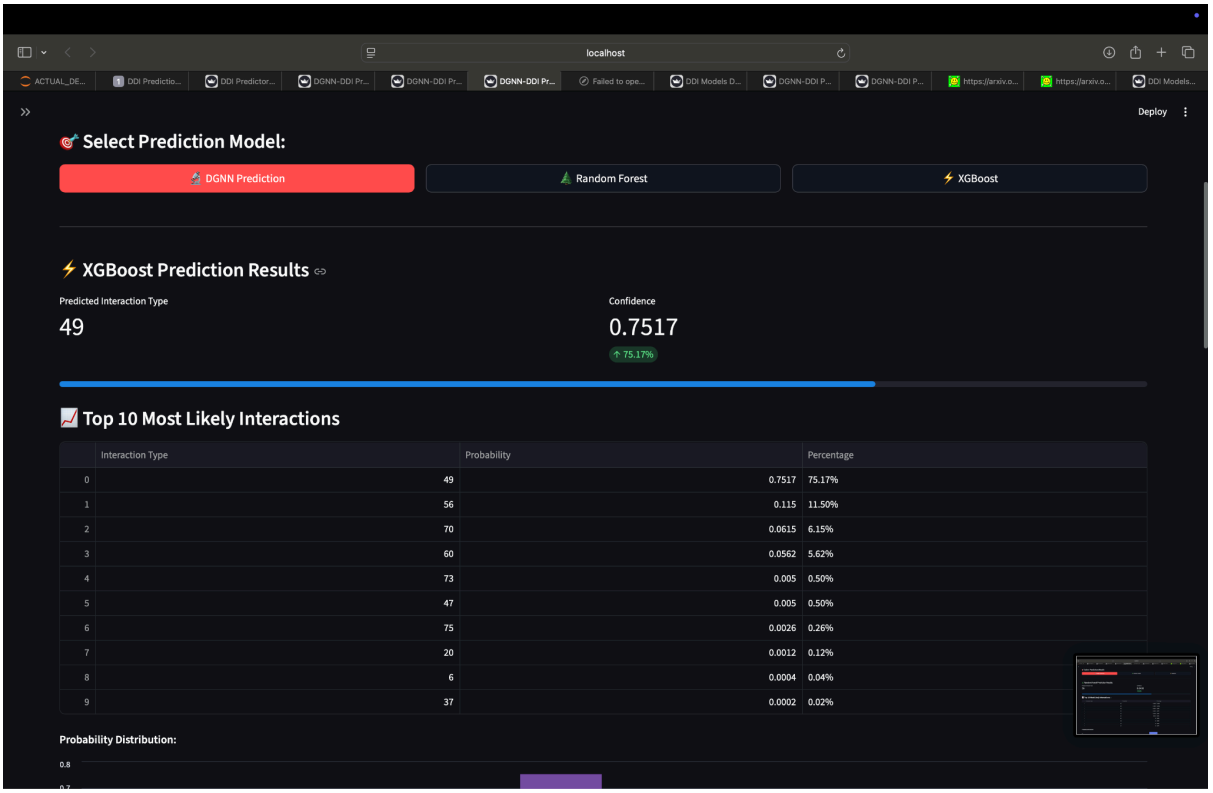
Streamlit app



GNN MODEL PREDICTION



RANDOM FOREST PREDICTION



XGBOOST PREDICTION

8.2 Deployment Strategy

The deployment lifecycle is organized into three phases: **pre-deployment validation**, **deployment execution**, and **post-deployment testing**.
Before deployment, the system undergoes rigorous validation to confirm the integrity of the three ML models, the correctness of **drug_data.pkl**, and the stability of all Python modules. This stage also verifies that RDKit is functioning correctly—a common failure point due to chemistry-related dependencies—and checks that ports 8501 and 8502 are available for Streamlit.

Pre-Deployment Validation Includes:

- Model deserialization (DGNN, RF, XGBoost)
- Validation of drugbank.csv, prediction_history.json, user_feedback.json
- Python syntax checks (`py_compile`)
- RDKit SMILES parsing validation

During deployment, applications are launched locally through Streamlit or deployed as **systemd services** for continuous operation. Running the services under isolated users

allows automatic restarts upon crashes, controlled privilege management, and increased reliability.

Production Execution Commands:

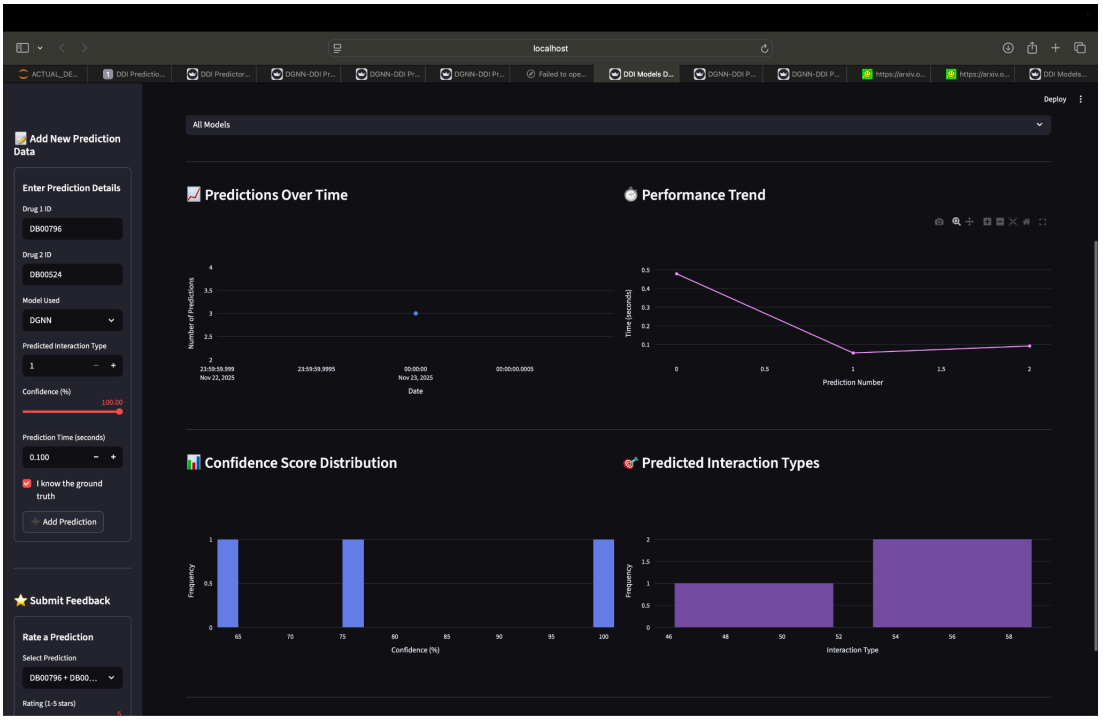
- `streamlit run app12.py --server.port=8501`
- `streamlit run dashboard.py --server.port=8502`

Post-deployment testing assesses startup stability, the dashboard's ability to read logs, and full integration between prediction and monitoring components.

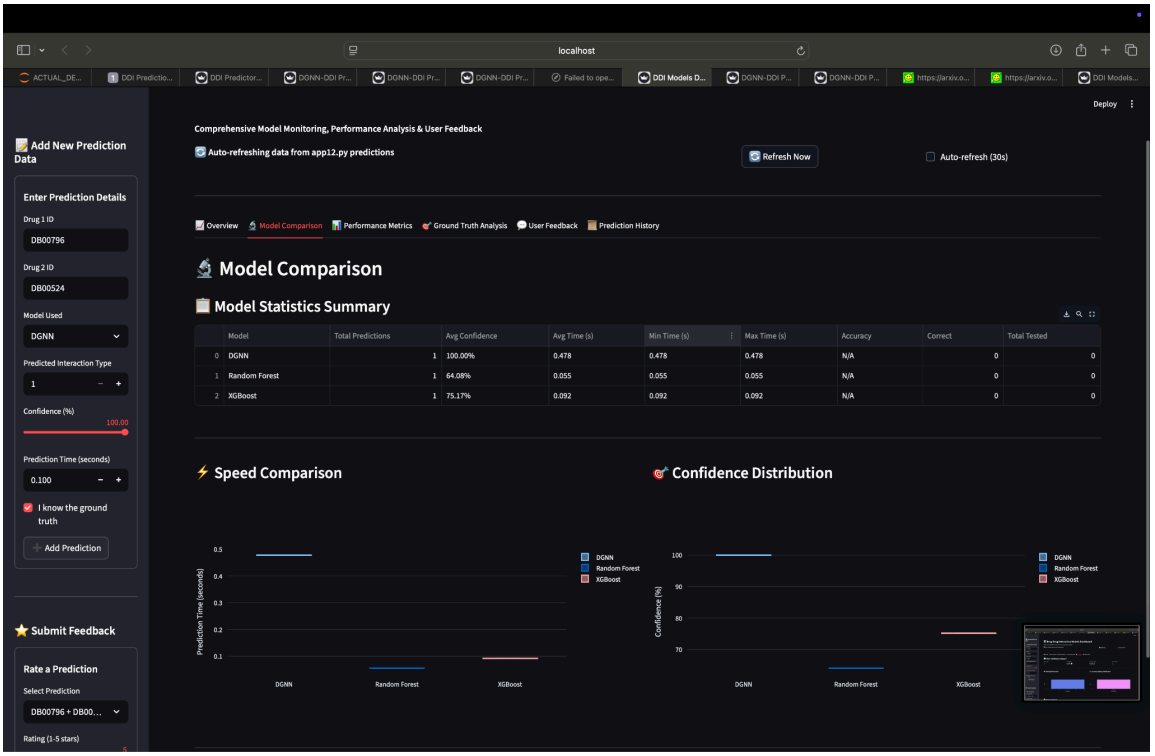
Performance Targets:

- DGNN prediction latency: **< 2 seconds**
- RF/XGBoost prediction latency: **< 1 second**
- Dashboard loading time: **< 3 seconds**
- Logging latency: **< 50 ms**

These thresholds ensure the system behaves reliably as a real-time clinical decision support tool.



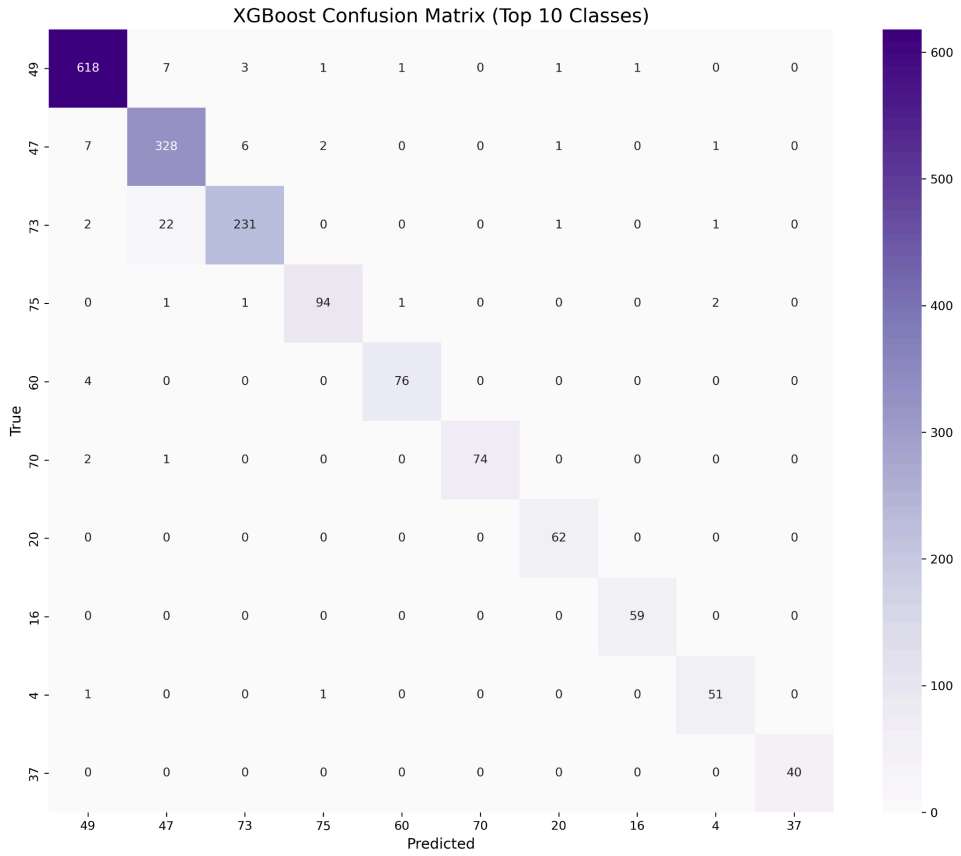
DASHBOARD.py

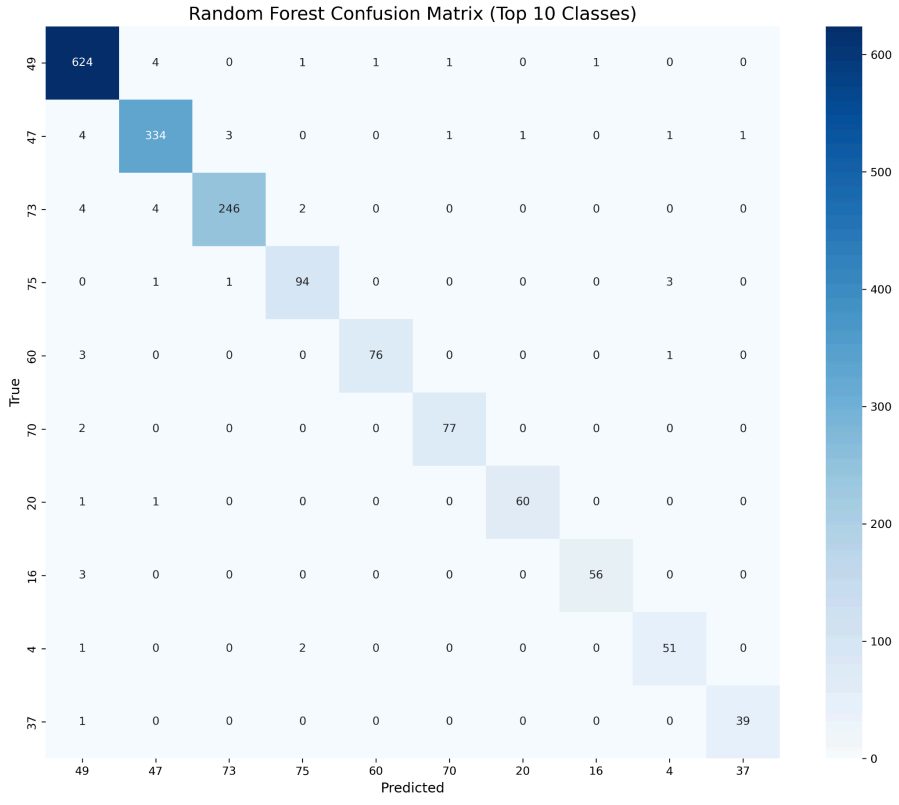


MODEL COMPARISON

Model	Test Accuracy	Macro Precision	Macro Recall	Macro F1-Score	Weighted Precision	Weighted Recall	Weighted F1-Score	Macro AUC
DGNN	0.9539	0.9450	0.9380	0.9415	0.9539	0.9539	0.9539	0.9850
Random Forest	0.9624812406203100	0.9876787362239820	0.9317500030991770	0.95263295979048	0.9631934105729640	0.9624812406203100	0.9618399618017550	0.9990552172700520
XGBoost	0.9564782391195600	0.9767565642262640	0.9558841391962430	0.9625394123569670	0.9570861617776360	0.9564782391195600	0.9560350898005720	0.9996045625365500

Model Comparison


XGBoost Confusion Matrix (Top 10 Classes)
True
Predicted
49 47 73 75 60 70 20 16 4 37
49 618 7 3 1 1 0 1 1 0 0
47 7 328 6 2 0 0 1 0 1 0
73 2 22 231 0 0 0 1 0 1 0
75 0 1 1 94 1 0 0 0 2 0
60 4 0 0 0 76 0 0 0 0 0
70 2 1 0 0 0 74 0 0 0 0
20 0 0 0 0 0 0 62 0 0 0
16 0 0 0 0 0 0 0 59 0 0
4 1 0 0 1 0 0 0 0 51 0
37 0 0 0 0 0 0 0 0 0 40
600 500 400 300 200 100 0


Random Forest Confusion Matrix (Top 10 Classes)
True
Predicted
49 47 73 75 60 70 20 16 4 37
49 624 4 0 1 1 1 0 1 0 0
47 4 334 3 0 0 1 1 0 1 1
73 4 4 246 2 0 0 0 0 0 0
75 0 1 1 94 0 0 0 0 3 0
60 3 0 0 0 76 0 0 0 1 0
70 2 0 0 0 0 77 0 0 0 0
20 1 1 0 0 0 0 60 0 0 0
16 3 0 0 0 0 0 0 56 0 0
4 1 0 0 2 0 0 0 0 51 0
37 1 0 0 0 0 0 0 0 0 39
600 500 400 300 200 100 0

9. Security, Trustworthiness & Risk Management

9.1 Security & Access Controls

Security mechanisms are integrated to prevent unauthorized access and ensure patient-sensitive data remains protected. Authentication is added using lightweight role-based credentials, while input validation uses strict RDKit-based SMILES checks to prevent malformed chemical structures or malicious inputs from crashing the models. Additionally, predictable failure handling ensures system stability even when corrupted model files or abnormal requests occur.

Security Mechanisms Used:

- Role-based authentication (researcher/admin)
- Hashed password protection
- Session expiration
- Strict SMILES validation
- DrugBank ID pattern checks
- Graceful model-loading error handling

9.2 Data Security & Privacy

All predictions and feedback are logged locally as JSON; however, the system is designed to support secure encrypted storage when scaled. Critical files—including model weights—use restricted Linux permissions to limit access. Daily log backups and rotation strategies ensure resilience while maintaining privacy. Audit logs capture detailed metadata surrounding every prediction.

Audit Log Captures:

- Timestamp
- Drug pair queried
- Model invoked
- Confidence score
- Prediction latency

This supports reproducibility, monitoring, and robust error tracing.

9.3 Risk Management

A structured risk-mitigation framework identifies key vulnerabilities and defines clear actions to reduce operational exposure.

Risk	Impact	Mitigation Strategy
Model corruption	High	Checksum validation; periodic load tests
Unauthorized access	High	Authentication; restricted file permissions
JSON log loss/corruption	Medium	Daily backup and log rotation
Slow model responses	High	Performance testing; caching where possible
Model drift	Medium	Regular evaluation; retraining using feedback

This approach ensures the system remains trustworthy and stable.

10. Evaluation, Monitoring & Maintenance Plan

10.1 System Evaluation & Monitoring

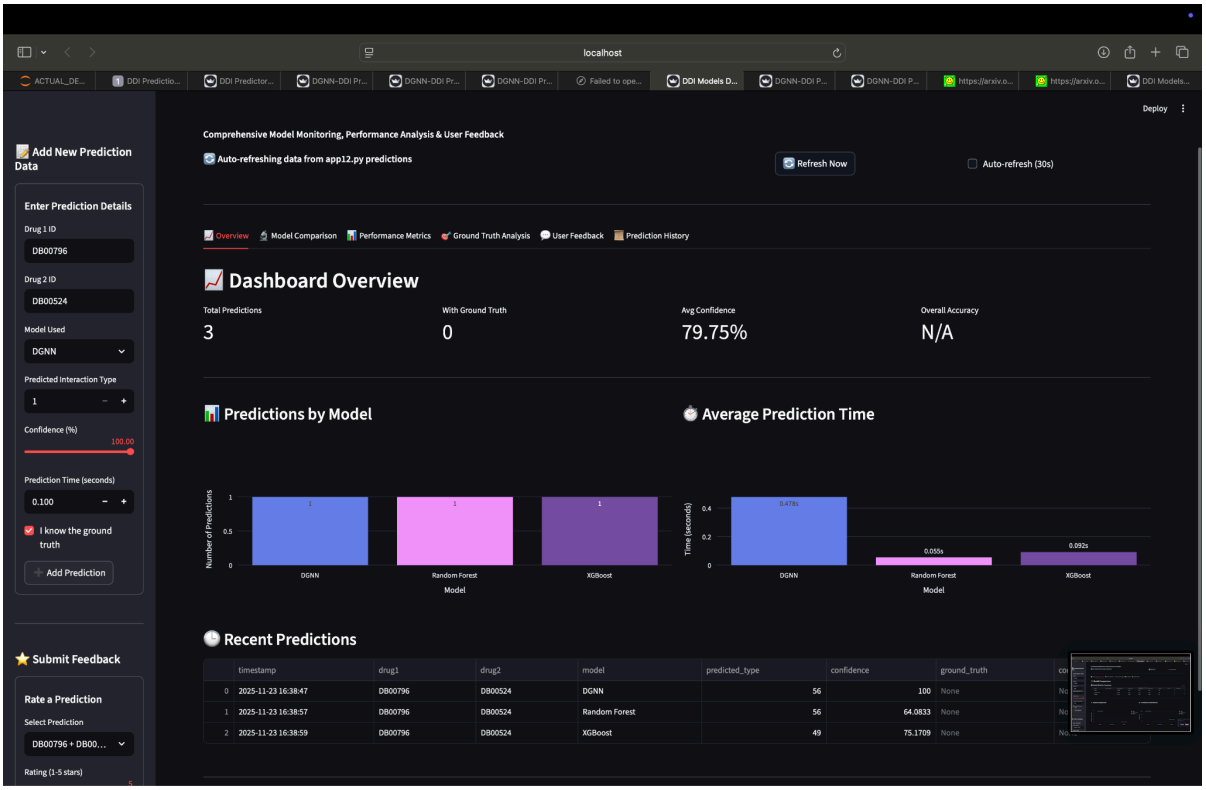
System monitoring is continuous and focuses on both **model performance** and **operational performance**.

Model metrics include accuracy trends, latency distributions, confidence levels, and prediction frequency across each ML model. Operational monitoring tracks application uptime, CPU and memory usage, disk consumption, and JSON log growth over time. The **dashboard.py** application aggregates all metrics into real-time visualizations, enabling rapid anomaly detection and system diagnostics.

Dashboard Tracks:

- Total predictions
- Confidence distribution
- Latency trends
- Predictions over time
- Most common interaction types

This creates a comprehensive evaluation loop for sustained system reliability.



MAIN_DASHBOARD

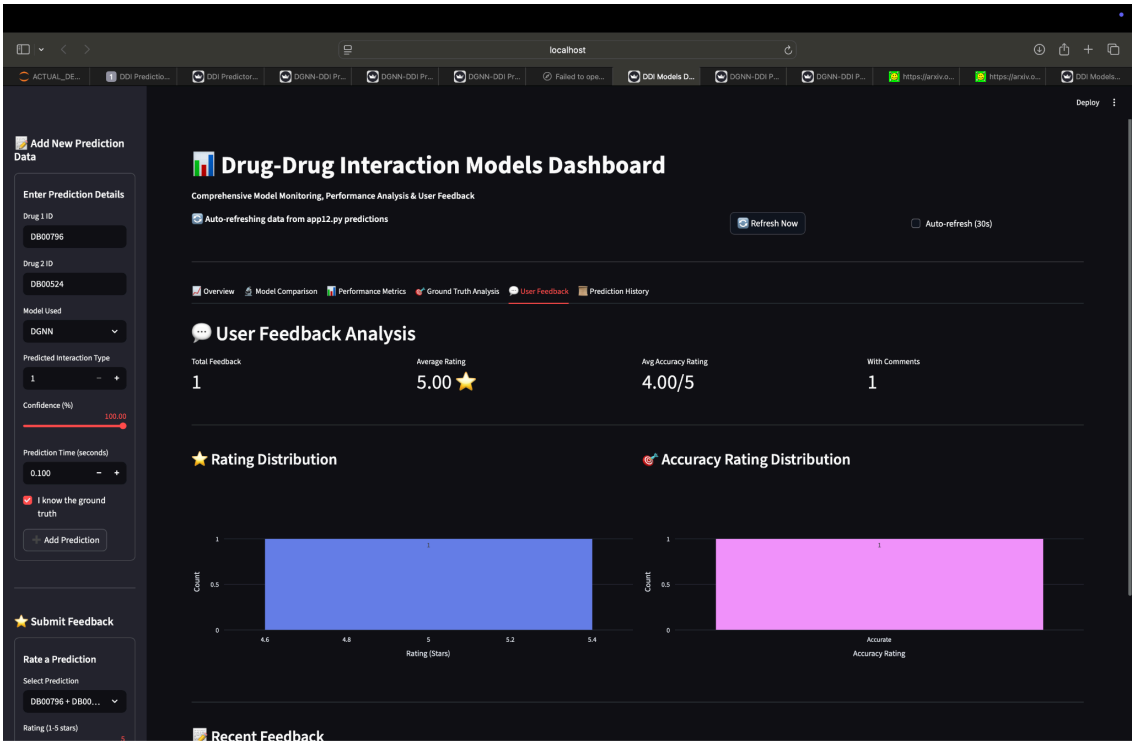
10.2 Feedback Collection & Continuous Improvement

The feedback module allows users to provide ratings and comments for individual predictions. These are utilized to evaluate model performance, detect systematic errors, and identify opportunities for retraining or refinement. Negative ratings automatically flag predictions for review.

Feedback KPIs Monitored:

- Average user rating
- Number of feedback submissions
- Model-wise rating distribution
- Recurring failure cases

This ensures that user insights directly contribute to model improvement cycles.



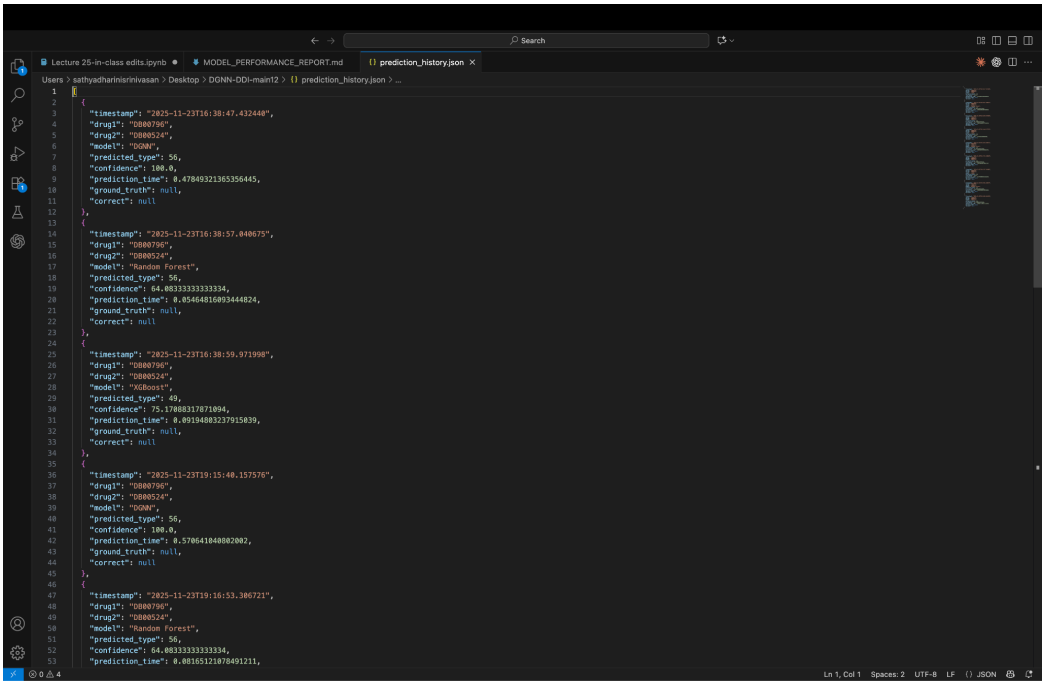
User Feedback.jpg

10.3 Maintenance & Compliance Audits

Maintenance operates on a structured recurring schedule to preserve long-term stability. Daily tasks include health monitoring and backups, while weekly routines analyze recent feedback and performance metrics. Monthly compliance audits review logging policies, file permissions, authentication strength, and overall model consistency with benchmark performance. Older logs (over 90 days) are purged as part of retention policies.

Maintenance Schedule:

- **Daily:** Check app health, CPU/memory usage, backup logs
- **Weekly:** Evaluate feedback, review accuracy trends
- **Monthly:** Full compliance audit, log archiving, documentation updates



MONTORING LOGS

These steps enforce responsible AI operation and ensure the system remains trustworthy, transparent, and compliant with best-practice ML governance.