IBM Innovation Centre for Education

PHEMESOFT Software That Matters

UPES

# University of Petroleum and Energy Studies

# Internship - High Level Design

# On

# Efficient Drug Discovery using Molecular Deep Learning

**Team Members:**

Sakshi Sehrawat (R2142210684)
Samyak Gupta (R2142210688)
Saksham (R2142210683)
Sadhashiva S (R2142210668)
Rohan Yadav (R2142210654)

**GUIDED BY :-**

Mr. Sumit Shukla

Industry Mentor:

# Table of Contents

# 1. Introduction

## 1.1 Scope of the Document

This document provides a comprehensive overview of the high-level design for the drug discovery system using molecular deep learning. It details the architecture, components, data design, interfaces, and non-functional requirements, ensuring a thorough understanding for all stakeholders involved in the project.

## 1.2 Intended Audience

This document is intended for:

- **Software Developers**: Responsible for implementing system components and modules.
- **Data Scientists**: Focused on developing and training machine learning models.
- **System Architects**: Tasked with designing the overall system architecture.
- **Project Managers**: Overseeing project development to ensure objectives are met.
- **Quality Assurance**: Ensuring the system components are tested and validated.
- **Regulatory Personnel**: Ensuring compliance with industry standards and regulations.

## 1.3 System Overview

The system leverages deep learning models to identify potential drug candidates and predict their toxicity. It integrates various datasets related to drug properties and toxicology, implementing both client-side and server-side validations. Key modules include data preprocessing, model training, model evaluation, and result visualization.

# 2. System Design

## 2.1 Application Design

The application is designed to be modular, scalable, and user-friendly, ensuring efficient handling of large datasets and complex computations. Key components include:

- **Frontend**: A user interface for data upload, model training, and result visualization.
- **Backend**: Handles data processing, model training, predictions, and database interactions.
- **Database**: Stores drug information, toxicology data, and model results.
- **APIs**: Facilitate communication between frontend and backend, and with external systems.

## 2.2 Process Flow

1. **User Uploads Data**: The user uploads datasets via the frontend.
2. **Data Validation**: Both client-side and server-side validations are performed.
3. **Data Preprocessing**: The backend preprocesses the data for model training.
4. **Model Training**: The preprocessed data is used to train the deep learning models.
5. **Predictions**: The trained models predict drug candidates and their toxicity.
6. **Result Visualization**: The predictions are displayed to the user on the frontend.

## 2.3 Information Flow

- **Data Input**: Users upload datasets (e.g., drug properties, toxicology data).
- **Processing**: Backend processes data and trains models.
- **Output**: Predictions and evaluations are stored in the database and displayed on the frontend.

## 2.4 Components Design

- **Frontend Components**:

  - **DataUploadComponent**: Handles data upload and validation.
  - **ModelTrainingComponent**: Manages model selection, parameter input, and training initiation.
  - **ResultsComponent**: Displays model predictions and evaluation metrics.
- **Backend Components**:
  - **DataProcessingService**: Cleans and preprocesses input data.
  - **ModelTrainingService**: Manages model training, evaluation, and saving trained models.
  - **PredictionService**: Handles model inference and returns predictions.

## 2.5 Key Design Considerations

- **Scalability**: Ensure the system can handle increasing amounts of data and users.
- **Modularity**: Design components to be reusable and easily maintainable.
- **Performance**: Optimize processing time and response rates.
- **Security**: Implement robust security measures for data protection.
- **Usability**: Ensure the system is user-friendly and accessible to non-technical users.

## 2.6 API Catalogue

- **Data Upload API**: Allows users to upload datasets.
- **Model Training API**: Initiates model training and returns status.
- **Prediction API**: Provides model predictions based on input data.
- **Results API**: Retrieves and displays prediction results.
- **Authentication API**: Manages user authentication and authorization.

# 3. Data Design

## 3.1 Data Model

The database schema includes tables for drug information, toxicology data, and model results.

- **Drug Information Table**: Stores data on drugs including name, properties, and known targets.
  - Fields: drug_id, drug_name, smiles, molecular_weight, targets.
- **Toxicology Data Table**: Contains toxicity information of various compounds.
  - Fields: compound_id, toxicity_level, in_vitro_results, in_vivo_results.
- **Model Results Table**: Stores the results of model predictions and evaluations.
  - Fields: result_id, drug_id, predicted_toxicity, prediction_date, model_version.

## 3.2 Data Access Mechanism

- **Admin**: Full access to all tables for data management and model configuration.
- **User**: Restricted access to view data and upload new datasets.
- **Guest**: Limited access to view public data and results.

## 3.3 Data Retention Policies

- **Data Retention**: Define policies for retaining and archiving data.
- **Backup**: Regular backups to prevent data loss.
- **Data Purging**: Remove outdated or irrelevant data periodically.

## 3.4 Data Migration

- **Migration Scripts**: Develop scripts for migrating data from legacy systems.
- **Validation**: Ensure data integrity during and after migration.
- **Testing**: Conduct thorough testing to validate migrated data.

# 4. Interfaces

The system interfaces with external APIs, model libraries, and visualization tools to ensure seamless integration and efficient operation. Detailed descriptions include:

## 4.1 External APIs

- **Drug and Toxicology Data APIs**: Integrate with external databases to fetch drug properties and toxicology data.
- **Authentication APIs**: Ensure secure user authentication and authorization processes.
- **Data Upload and Retrieval APIs**: Facilitate data upload and retrieval from external sources.

## 4.2 Model Libraries

- **TensorFlow/PyTorch**: Utilize these libraries for implementing and training deep learning models.
- **Scikit-learn**: Implement additional machine learning algorithms and preprocessing tools.

## 4.3 Visualization Tools

- **Matplotlib**: Generate static, animated, and interactive visualizations.
- **Seaborn**: Create informative and attractive statistical graphics.
- **Plotly/D3.js**: Develop interactive web-based visualizations for model predictions and evaluations.

# 5. State and Session Management

## 5.1 Session Management

- **Flask/Django Sessions**: Manage user sessions using Flask or Django frameworks.
- **Session Storage**: Securely store session data with appropriate expiration policies.
- **JWT (JSON Web Tokens)**: Implement JWT for secure session handling and token-based authentication.

## 5.2 User Roles and Permissions

- **Admin Role**: Full access to all system features and data.
- **User Role**: Restricted access to data upload and viewing of results.
- **Guest Role**: Limited access to viewing public data and results.

# 6. Caching

## 6.1 Caching Mechanism

- **Redis**: Implement Redis for efficient caching of frequently accessed data and model results.
- **In-Memory Caching**: Store frequently used data in memory to reduce database load and improve performance.

## 6.2 Cache Strategies

- **Time-Based Expiration**: Set time-based expiration policies for cached data to ensure freshness.
- **Cache Invalidation**: Implement strategies to invalidate and update the cache when underlying data changes.

# 7. Non-Functional Requirements

## 7.1 Security Aspects

- **Data Encryption**: Encrypt sensitive data in transit and at rest to protect against unauthorized access.
- **Access Control**: Implement role-based access control to restrict data access based on user roles.
- **Monitoring and Logging**: Continuous monitoring and logging of system activities to detect and respond to security incidents.
- **Regulatory Compliance**: Ensure compliance with data privacy regulations (e.g., GDPR) and industry standards.

## 7.2 Performance Aspects

- **Scalability**: Design the system to handle increased data volumes and user loads by scaling horizontally and vertically.
- **Optimization**: Optimize algorithms, database queries, and data processing to enhance performance.
- **Load Balancing**: Distribute workloads evenly across multiple servers to prevent bottlenecks and ensure high availability.
- **Response Time**: Ensure the system responds promptly to user requests, aiming for low latency and high throughput.

# 8. References

1. Gawehn E, Hiss JA, Schneider G. Deep Learning in Drug Discovery. Mol Inform. 2016 Jan;35(1):3-14. doi: 10.1002/minf.201501008. Epub 2015 Dec 30. PMID: 27491648.

2. Kim J, Park S, Min D, Kim W. Comprehensive Survey of Recent Drug Discovery Using Deep Learning. International Journal of Molecular Sciences. 2021; 22(18):9983. https://doi.org/10.3390/ijms22189983

3. Askr, H., Elgeldawi, E., Aboul Ella, H. et al. Deep learning in drug discovery: an integrative review and future challenges. Artif Intell Rev 56, 5975–6037 (2023). https://doi.org/10.1007/s10462-022-10306-1

4. DeepChem. (n.d.). DeepChem: A Python library for deep learning in chemistry and biology from https://www.ijedr.org/papers/IJEDR2104012.pdf