IBM Innovation Centre for Education

PHEMESOFT Software That Matters

UPES

University of Petroleum and Energy Studies

Internship - Low 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

The quest to discover new drugs is a complex and often time-consuming process, involving multiple steps from the initial identification of potential compounds to rigorous testing and regulatory approval. Traditional methods, while effective, are often slow, labor-intensive, and costly. However, the advent of deep learning, a subset of artificial intelligence (AI), is revolutionising the field of drug discovery. Deep learning promises to accelerate the drug discovery process, reduce costs, and improve the success rate of identifying viable drug candidates.

Deep learning involves the use of artificial neural networks with multiple layers that can learn from large volumes of data. These models can identify patterns, make predictions, and even generate new hypotheses based on the data they process. In the context of drug discovery, deep learning models are applied to analyze and interpret complex biological data, which includes genomic data, chemical structures, and clinical trial results. This approach enables researchers to predict the behavior of drug compounds, identify potential targets, and design new molecules with desired therapeutic properties.

1.1. Scope of the Document

This document outlines the technical details and design specifications required for developing the project. It covers the system architecture, components design, data design, and integration with other systems. It provides a detailed guide for software developers and data scientists to build, validate, and deploy the drug discovery system using molecular deep learning techniques.

1.2. Intended Audience

This document is intended for:

- **Software Developers:** who will implement the system components.
- **Data Scientists:** who will develop and train the machine learning models.
- **System Architects:** who will design the overall system architecture.
- **Project Managers:** who will oversee the project development and ensure it meets the objectives.

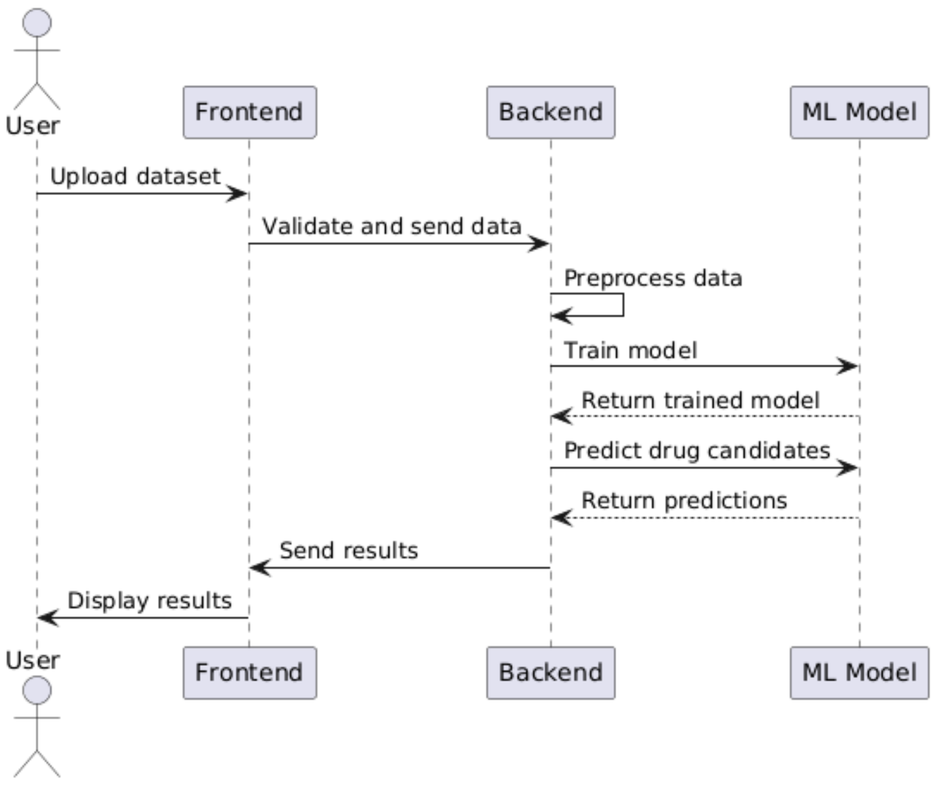
1.3. System Overview

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 and implements both client-side and server-side validations. The system includes modules for data preprocessing, model training, model evaluation, and visualization of results.

2. Low Level System Design

2.1. Sequence Diagram

The sequence diagram illustrates the interactions between different system components during the drug discovery process.



Description:

1. User initiates the process by uploading the dataset.
2. Frontend System validates the input data and sends it to the backend.
3. Backend System preprocesses the data.
4. Machine Learning Model is trained using the preprocessed data.
5. The trained model predicts the drug candidates and evaluates their toxicity.

6. Results are sent back to the Frontend System for visualization.
7. User views the results and makes decisions based on the predictions.

2.2. Navigation Flow/UI Implementation

The navigation flow includes:

Home Screen:

- **Components:** Navigation bar, project overview, quick links to main features.
- **Attributes:** User-friendly interface with clear navigation paths.

Data Upload:

- **Components:** File input, data preview, and upload button.
- **Attributes:** Validation for file formats (e.g., CSV), display sample data format.

Model Training:

- **Components:** Model selection dropdown, parameter input fields, start button.
- **Attributes:** Predefined hyperparameters, tooltips for parameter descriptions.

Results:

- **Components:** Tables, charts, and graphs for displaying predictions and metrics.
- **Attributes:** Interactive visualizations using libraries like D3.js or Plotly.

2.3. Screen Validations, Defaults, and Attributes

Data Upload Screen:

- **Validations:** Check file format (e.g., CSV, JSON), file size limit (e.g., 100MB), mandatory fields.
- **Defaults:** Provide sample data format and default values where applicable.
- **Attributes:** File input control, data preview area, and upload button.

Model Training Screen:

- **Validations:** Validate model parameters (e.g., learning rate, epochs) and dataset selection.
- **Defaults:** Default hyperparameters for models based on best practices.
- **Attributes:** Model selection dropdown, parameter input fields, and start training button.

2.4. Client-Side Validation Implementation

- Use JavaScript/TypeScript for client-side validation.
- Validate data inputs (e.g., file formats, numerical ranges) before submission.
- Provide real-time feedback to the user on validation errors using libraries like Formik and Yup.

2.5. Server-Side Validation Implementation

- Implement validation checks in the backend using Python frameworks like Flask or Django.
- Ensure data integrity and security by validating inputs before processing.
- Log validation errors for monitoring and debugging.

2.6. Components Design Implementation

Frontend:

- Use React.js for a responsive and interactive user interface.
- Implement reusable components (e.g., forms, tables, charts).

Backend:

- Use Flask/Django for RESTful API development.
- Implement services for data processing, model training, and predictions.

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.

2.7. Configurations/Settings

Database Configuration:

- Setup and configure PostgreSQL/MySQL database.

Model Configuration:

- Define hyperparameters and training settings.

Environment Configuration:

- Use environment variables for sensitive settings (e.g., API keys).

Configuration Files:

- **config.yaml:** Stores configuration settings for the application.
- **.env:** Contains environment-specific variables.

2.8. Interfaces to Other Components

Data Sources: Integrate with external databases and APIs for drug and toxicology data. **Model Libraries:** Use TensorFlow/PyTorch for model implementation. **Visualization Tools:** Integrate with libraries like Matplotlib and Seaborn for result visualization.

3. Data Design

3.1. List of Key Schemas/Tables in Database

Drug Information: Stores data on drugs including name, properties, and known targets.

- **Fields:** drug_id, drug_name, smiles, molecular_weight, targets

Toxicology Data: Contains toxicity information of various compounds.

- **Fields:** compound_id, toxicity_level, in_vitro_results, in_vivo_results

Model Results: Stores the results of model predictions and evaluations.

- **Fields:** result_id, drug_id, predicted_toxicity, prediction_date, model_version

3.2. Details of Access Levels on Key Tables in Scope

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. Key Design Considerations in Data Design

- Ensure data normalization to reduce redundancy.
- Implement indexing for faster query performance.
- Define relationships and foreign keys for data integrity.

4. Details of Other Frameworks Being Used

4.1. Session Management

- Use Flask/Django session management for user authentication and authorization.
- Store session data securely with expiration policies.
- Implement JWT (JSON Web Tokens) for secure session handling.

4.2. Caching

- Implement caching using Redis for faster data retrieval.
- Cache frequently accessed data and model results to enhance performance.
- Use caching strategies like time-based expiration and invalidation on data updates.

5. Unit Testing

- Write unit tests for each component using pytest/unittest.
- Test data preprocessing, model training, and prediction functionalities.
- Ensure coverage of edge cases and error handling.

6. Key Notes

- Ensure compliance with data privacy regulations (e.g., GDPR).
- Maintain clear documentation for all modules and functions.
- Follow best practices for coding, testing, and deployment.
- Regularly update the models and retrain them with new data to improve accuracy.

7. 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