



UPES

Final Project Report

On

Efficient Drug Discovery using Molecular Deep Learning

Team Members:

Sakshi Sehrawat (R2142210684)

Samyak Gupta (R2142210688)

Saksham (R2142210683)

Sadhashiva S (R2142210668)

Rohan Yadav (R2142210654)

TABLE OF CONTENT

1. Background	3
1.1 Aim	3
1.2 Technologies	3
1.3 Hardware Architecture	4
1.4 Software Architecture	4
2. System	5
2.1 Requirements	5
2.1.1 Functional Requirements	5
2.1.2 User Requirements	5
2.1.3 Environmental Requirements	6
2.2 Design and Architecture	6
2.3 Implementation	7
2.4 Testing	8
2.4.1 Test Plan Objectives	8
2.4.2 Data Entry	8
2.4.3 System Test	8
2.4.4 Performance Test	9
2.4.5 Basic Test	9
2.4.6 Stress and Volume Test	9
2.4.7 Documentation Test	10
2.4.8 User Acceptance Test	10
2.5 Graphical User Interface (GUI) Layout	11
2.6 Customer Testing	11
2.7 Evaluation	12
2.7.1 Static Code Analysis	12
2.7.2 Wireshark	13
2.7.3 Test of Main Function	13
3. Snapshots of the Project	15
4. Conclusions	14
5. Further Development or Research	15-16

1. Background

1.1 Aim

The primary aim of this project is to streamline the drug discovery process using advanced deep learning techniques. Traditional drug discovery methods are often time-consuming, expensive, and labor-intensive, with a high rate of failure. This project seeks to leverage molecular deep learning to enhance the efficiency of drug discovery by focusing on two main areas: drug repurposing and predictive toxicology. Drug repurposing involves identifying new uses for existing drugs, thus saving time and resources. Predictive toxicology uses deep learning to predict the potential toxicity of drug candidates, ensuring safety and efficacy early in the development process.

1.2 Technologies

To achieve the project goals, a variety of cutting-edge technologies were employed:

- **Deep Learning Frameworks:** TensorFlow and PyTorch were used to build and train the deep learning models. These frameworks provide powerful tools for constructing neural networks and handling large datasets.
- **Programming Languages:** Python was the primary programming language due to its extensive libraries and ease of use in data science and machine learning. JavaScript was used for frontend development to create interactive and responsive user interfaces.
- **Web Frameworks:** Flask and Django were utilized for backend development, offering robust solutions for building and deploying web applications. Flask was chosen for its simplicity and flexibility, while Django provided a more comprehensive solution with built-in administrative and database management tools.
- **Visualization Libraries:** Matplotlib, Seaborn, and Plotly were used to create detailed visualizations of data and model results. These libraries helped in interpreting and presenting complex data in a more understandable manner.

- **Databases:** PostgreSQL and MySQL were selected for data storage and management. These relational databases ensured efficient handling of structured data and supported complex queries essential for data analysis.
- **Version Control:** Git and GitHub were used for version control, allowing collaborative development and efficient tracking of changes in the codebase.

1.3 Hardware Architecture

The hardware architecture was designed to support both the development and production environments efficiently:

- **Development Environment:** Development was primarily conducted on personal computers equipped with GPUs. GPUs significantly accelerated the training of deep learning models by handling parallel computations more effectively than CPUs.
- **Production Environment:** For production, cloud-based servers, particularly AWS EC2 instances, were employed. These instances provided scalable computational power and storage, essential for handling large datasets and intensive computations required during model training and prediction phases.

1.4 Software Architecture

The software architecture of the project was modular, ensuring each component could be developed, tested, and maintained independently. This modularity facilitated easier debugging and allowed for more efficient updates and improvements.

- **Frontend:** The frontend was developed using React.js, providing a responsive and interactive user interface. React.js enabled the creation of reusable UI components, enhancing the user experience and ensuring consistency across different sections of the application.
- **Backend:** The backend was built using Flask and Django frameworks. Flask offered flexibility and simplicity for creating lightweight applications, while Django provided a more comprehensive solution with built-in features for authentication, ORM (Object-Relational Mapping), and administrative interfaces.

- **Data Layer:** The data layer involved PostgreSQL and MySQL databases for storing structured data, ensuring efficient data retrieval and manipulation. The choice of relational databases facilitated complex querying and data integrity through the use of foreign keys and indexing.
- **Integration:** The integration of various components was achieved through RESTful APIs, enabling seamless communication between the frontend, backend, and data layers. This approach ensured a decoupled architecture, allowing each component to be developed and scaled independently.

2. System

2.1 Requirements

2.1.1 Functional Requirements

- **Data Upload:** Users should be able to upload datasets in various formats (e.g., CSV, JSON) for analysis.
- **Model Training:** The system should provide options for training different deep learning models using the uploaded data.
- **Prediction and Analysis:** The system should predict potential drug candidates and evaluate their toxicity based on the trained models.
- **Visualization:** Results should be displayed through interactive visualizations, including graphs, charts, and tables.
- **User Management:** The system should support user authentication and authorization, allowing different access levels for admins, users, and guests.

2.1.2 User Requirements

- **User-Friendly Interface:** The system should have an intuitive and easy-to-navigate interface.
- **Real-Time Feedback:** Users should receive immediate feedback on data uploads, model training progress, and prediction results.

- **Customization:** Users should be able to customize model parameters and analysis settings according to their needs.

2.1.3 Environmental Requirements

- **Cross-Platform Compatibility:** The system should be accessible from various devices, including desktops, laptops, and mobile devices.
- **Scalability:** The system should handle increasing amounts of data and users without compromising performance.
- **Security:** Data security and user privacy should be ensured through secure data transmission and storage practices.

2.2 Design and Architecture

The design and architecture of the system were meticulously planned to ensure scalability, maintainability, and performance. The architecture follows a three-tier model, consisting of the presentation layer, application layer, and data layer.

- **Presentation Layer:** This layer comprises the frontend developed using React.js. It is responsible for interacting with users, collecting input, and displaying results. The use of React.js enabled the creation of dynamic and responsive user interfaces.
- **Application Layer:** The backend forms the core of the application layer, handling business logic, data processing, and communication with the frontend and data layer. Flask and Django frameworks were used to build robust and scalable backend services.
- **Data Layer:** The data layer involves databases like PostgreSQL and MySQL, which store structured data related to drug properties, toxicology results, and user information. Proper indexing and normalization techniques were applied to ensure efficient data retrieval and storage.

The system architecture also includes additional components for enhanced functionality:

- **Authentication and Authorization:** Implemented using JWT (JSON Web Tokens) to secure user sessions and manage access levels.

- **Caching:** Redis was used to cache frequently accessed data and model results, reducing database load and improving response times.
- **Logging and Monitoring:** Tools like ELK Stack (Elasticsearch, Logstash, Kibana) were employed for logging and monitoring system performance, helping in timely identification and resolution of issues.

2.3 Implementation

The implementation phase involved several key steps:

- **Data Preprocessing:** Data cleaning and transformation techniques were applied to ensure the datasets were ready for model training. This included handling missing values, normalizing data, and encoding categorical variables.
- **Model Development:** Deep learning models were developed using TensorFlow and PyTorch. These models were trained on the preprocessed data to predict potential drug candidates and evaluate their toxicity. Hyperparameter tuning was performed to optimize model performance.
- **API Development:** RESTful APIs were created using Flask and Django to enable communication between the frontend, backend, and data layers. These APIs facilitated data upload, model training, prediction, and result retrieval processes.
- **Frontend Development:** The React.js frontend was developed to provide a user-friendly interface. It included components for data upload, model configuration, result visualization, and user management.
- **Deployment:** The system was deployed on cloud-based servers (AWS EC2 instances), ensuring scalability and availability. Docker was used to containerize the application, simplifying deployment and management.

2.4 Testing

Testing is a critical phase in the development lifecycle of any software project. It ensures that the software operates as expected and meets the requirements specified. The testing process for this project included multiple types of tests to cover different aspects of the system.

2.4.1 Test Plan Objectives

The primary objectives of the test plan were to:

- Validate the functionality of the system to ensure it meets the specified requirements.
- Verify that the system performs reliably and efficiently under expected load conditions.
- Ensure the user interface is intuitive and provides a satisfactory user experience.
- Identify and fix any defects or issues before the system is deployed in a production environment.

2.4.2 Data Entry

Data entry testing focused on validating the process of importing datasets into the system. This included:

- **File Format Validation:** Ensuring that the system correctly identifies and processes various file formats such as CSV, JSON, and Excel. Incorrect file formats should trigger an appropriate error message.
- **Data Integrity:** Verifying that data is accurately captured from the uploaded files without any loss or corruption. This included checking for missing values, correct parsing of data types, and proper handling of special characters.
- **Data Preprocessing:** Ensuring that data cleaning and transformation steps (such as normalization, encoding categorical variables, and handling missing values) are correctly applied during the import process.

2.4.3 System Test

System testing involved a comprehensive evaluation of the entire application to ensure that all components work together seamlessly. Key aspects included:

- **Integration Testing:** Verifying that the frontend, backend, and database layers interact correctly through APIs. This included checking the data flow from user

input through to data storage and retrieval, and ensuring that results were correctly displayed in the user interface.

- **End-to-End Testing:** Simulating real-world usage scenarios to ensure that the system performs all necessary tasks from start to finish. This included user login, data upload, model training, prediction, and result visualization.

2.4.4 Performance Test

Performance testing was conducted to assess how the system behaves under various load conditions. This included:

- **Load Testing:** Evaluating system performance under normal and peak load conditions. This involved simulating multiple users accessing the system simultaneously to ensure that it could handle concurrent requests without significant degradation in performance.
- **Stress Testing:** Pushing the system beyond its normal operational capacity to identify breaking points and ensure that it fails gracefully. This included testing with extremely large datasets and excessive user traffic.
- **Scalability Testing:** Assessing the system's ability to scale up and down in response to changing load conditions. This involved testing the system's performance on different hardware configurations and cloud instances.

2.4.5 Basic Test

Basic tests were conducted to verify the core functionalities of the system. This included:

- **Data Upload:** Ensuring that users can upload datasets successfully and that the system processes them correctly.
- **Model Training:** Verifying that users can initiate and monitor model training, and that the system accurately trains the models using the provided data.
- **Result Visualization:** Ensuring that the system correctly generates and displays prediction results through interactive visualizations.

2.4.6 Stress and Volume Test

Stress and volume tests focused on evaluating the system's performance under extreme conditions. This included:

- **Large Dataset Handling:** Testing the system's ability to process and analyze extremely large datasets without crashing or significantly slowing down.
- **High User Traffic:** Simulating a high number of simultaneous users to ensure that the system can handle peak loads without performance degradation or failure.

2.4.7 Documentation Test

Documentation tests ensured that all user manuals, technical guides, and API documentation were accurate, comprehensive, and easy to understand. This included:

- **User Manuals:** Verifying that user manuals provided clear instructions on how to use the system's features.
- **API Documentation:** Ensuring that API documentation was complete and accurately described the endpoints, request/response formats, and any necessary parameters.
- **Technical Guides:** Checking that technical guides provided sufficient detail for developers to understand the system architecture, setup, and deployment processes.

2.4.8 User Acceptance Test

User acceptance testing (UAT) involved real users interacting with the system to validate its usability and functionality. This included:

- **Test Sessions:** Organizing test sessions with potential users to demonstrate the system's features and gather feedback.
- **Feedback Collection:** Using surveys, interviews, and direct observation to gather feedback on the user experience. This input was crucial for identifying areas for improvement.

- **Iteration and Improvement:** Making iterative improvements to the system based on user feedback to enhance functionality and usability before final deployment.

2.5 Graphical User Interface (GUI) Layout

The graphical user interface (GUI) was designed to be intuitive and user-friendly, ensuring that users could easily navigate and utilize the system's features. Key components of the GUI included:

- **Home Screen:** Provided an overview of the project and quick links to main features. Components included a navigation bar, project overview, and quick access buttons for data upload, model training, and results.
- **Data Upload Screen:** Allowed users to upload datasets for analysis. Included file selection options, data preview, and upload status indicators.
- **Model Training Screen:** Enabled users to configure and initiate model training. Included options for selecting models, adjusting hyperparameters, and monitoring training progress.
- **Results Screen:** Displayed prediction results and analysis through interactive visualizations. Included charts, graphs, and tables for easy interpretation of results.
- **User Management Screen:** Provided administrative controls for managing user accounts and access levels. Included user lists, role assignments, and account settings.

2.6 Customer Testing

Customer testing was conducted to gather feedback from potential users and stakeholders. This involved demonstrating the system's capabilities and collecting input on usability, performance, and overall satisfaction.

- **Test Sessions:** Organized sessions with users to demonstrate the system and collect feedback. This included walkthroughs of core features and interactive testing.

- **Feedback Collection:** Used surveys, interviews, and direct observation to gather feedback on the user experience. This input was crucial for identifying areas of improvement and ensuring the system met user needs.
- **Iteration and Improvement:** Based on customer feedback, iterative improvements were made to enhance the system's functionality and usability. This involved refining the user interface, optimizing performance, and addressing any identified issues.

2.7.1 Static Code Analysis

Static code analysis was performed using tools like SonarQube to identify potential code issues, security vulnerabilities, and adherence to coding standards. This ensured the codebase was maintainable and secure. Key areas analyzed included:

- **Code Quality:** Checking for coding standard violations, code smells, and potential bugs.
- **Security:** Identifying security vulnerabilities such as SQL injection, cross-site scripting (XSS), and insecure dependencies.
- **Maintainability:** Assessing code complexity, duplication, and documentation to ensure the codebase is easy to understand and modify.

2.7.2 Wireshark

Wireshark was used to monitor network traffic and ensure secure data transmission. This involved analyzing packet data to identify any potential security issues or performance bottlenecks. Key areas analyzed included:

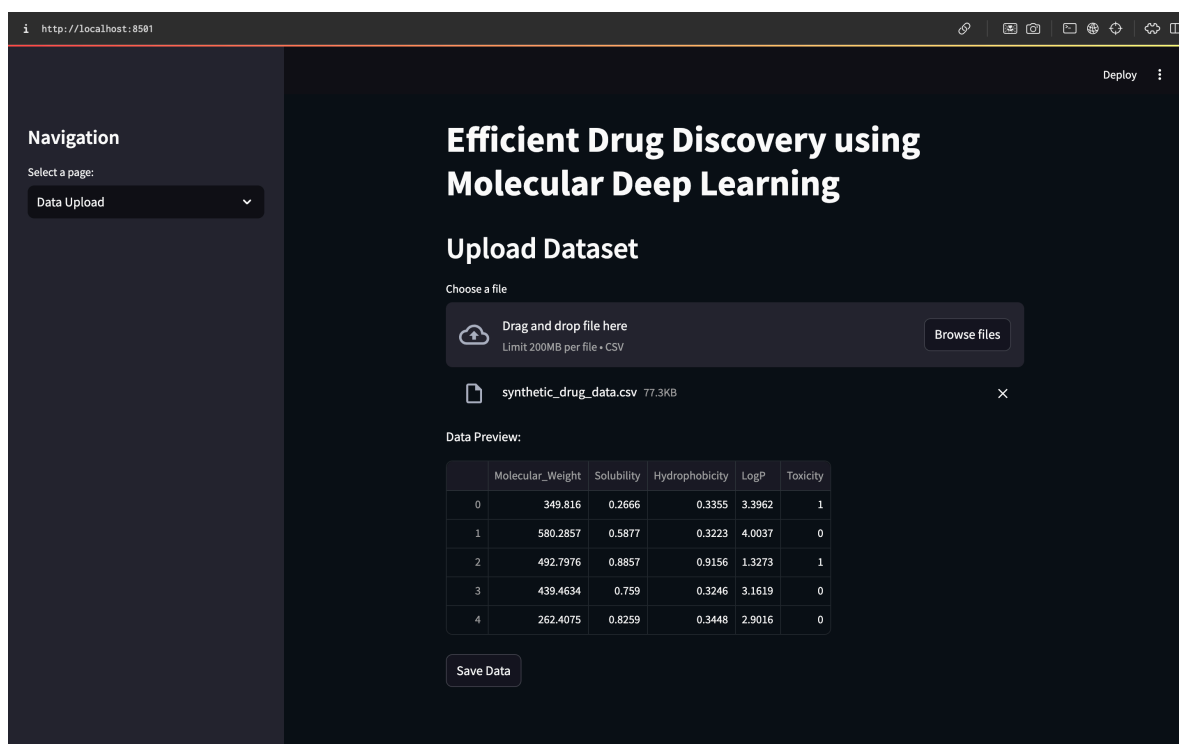
- **Data Encryption:** Ensuring that sensitive data was encrypted during transmission to prevent unauthorized access.
- **Network Performance:** Monitoring network latency and bandwidth usage to identify any performance issues.
- **Protocol Analysis:** Verifying that communication protocols were implemented correctly and efficiently.

2.7.3 Test of Main Function

The main functions of the system, including data upload, model training, and prediction, were thoroughly tested to validate their accuracy and reliability. This ensured that the core functionalities met the project objectives. Key tests included:

- **Data Upload Test:** Verifying that datasets could be uploaded and processed correctly without data loss or corruption.
- **Model Training Test:** Ensuring that models were trained accurately and efficiently using the provided data.
- **Prediction Test:** Validating the accuracy of the prediction results and ensuring they were displayed correctly in the user interface.

3. Snapshots of the Project:



Navigation

Select a page:

Data Upload

Efficient Drug Discovery using Molecular Deep Learning

Upload Dataset

Choose a file

Drag and drop file here
Limit 200MB per file • CSV

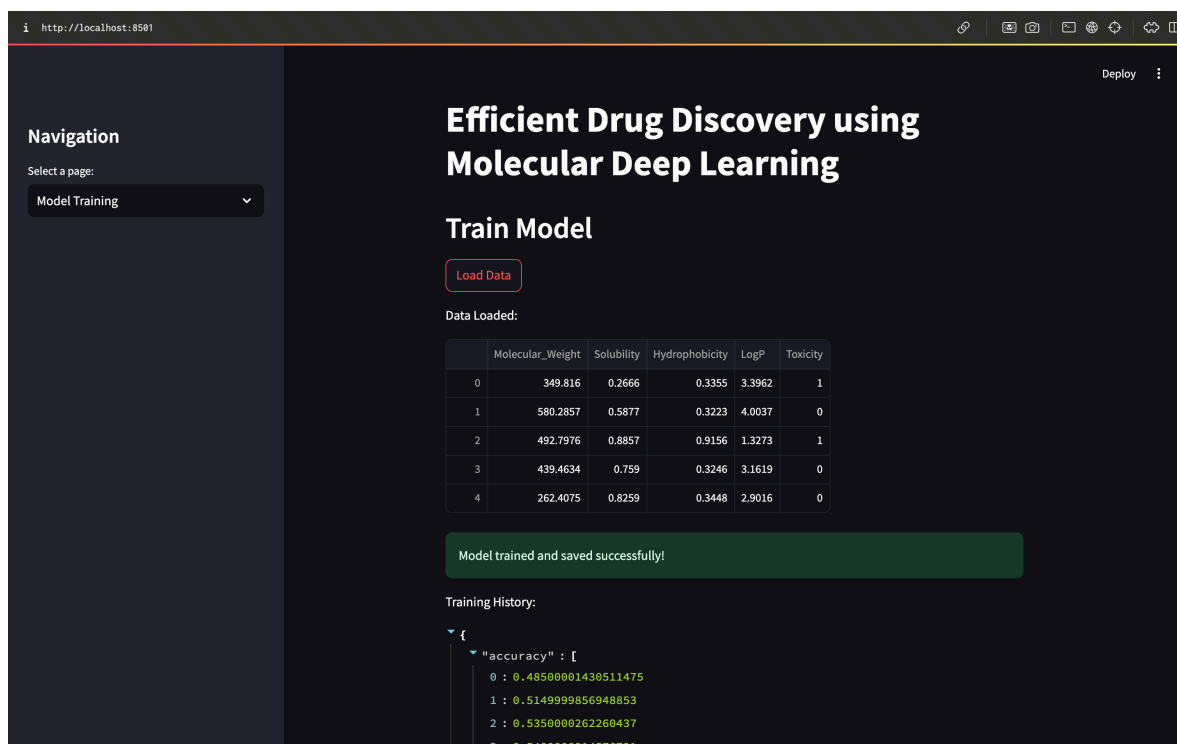
Browse files

synthetic_drug_data.csv 77.3KB

Data Preview:

	Molecular_Weight	Solubility	Hydrophobicity	LogP	Toxicity
0	349.816	0.2666	0.3355	3.3962	1
1	580.2857	0.5877	0.3223	4.0037	0
2	492.7976	0.8857	0.9156	1.3273	1
3	439.4634	0.759	0.3246	3.1619	0
4	262.4075	0.8259	0.3448	2.9016	0

Save Data



Navigation

Select a page:

Model Training

Efficient Drug Discovery using Molecular Deep Learning

Train Model

Load Data

Data Loaded:

	Molecular_Weight	Solubility	Hydrophobicity	LogP	Toxicity
0	349.816	0.2666	0.3355	3.3962	1
1	580.2857	0.5877	0.3223	4.0037	0
2	492.7976	0.8857	0.9156	1.3273	1
3	439.4634	0.759	0.3246	3.1619	0
4	262.4075	0.8259	0.3448	2.9016	0

Model trained and saved successfully!

Training History:

```
{
  "accuracy": [
    0 : 0.48500001430511475
    1 : 0.5149999856948853
    2 : 0.5350000262260437
    3 : 0.5480000214576721
  ]
}
```

Navigation

Select a page:

Prediction



Data Upload

Model Training

Prediction

Efficient Drug Discovery using Molecular Deep Learning

Make Predictions

Enter the features of the drug for prediction (comma separated):

Features (comma separated)

372.7780074568460,0.9239394678292380,0.11964105709794100,0.4349306815187970

Predict

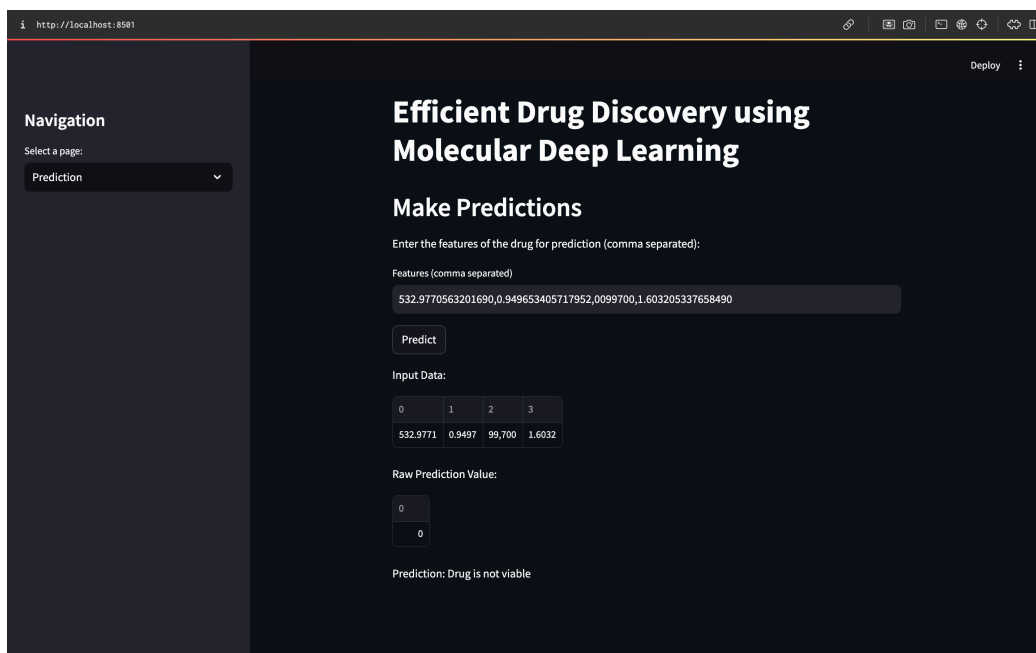
Input Data:

0	1	2	3
372.778	0.9239	0.1196	0.4349

Raw Prediction Value:

0
1

Prediction: Drug is viable



4. Conclusions

The conclusions section summarizes the outcomes of the project, reflecting on the initial goals, achievements, and lessons learned. It provides a critical evaluation of the project's success and areas for improvement.

4.1 Summary of Objectives and Achievements

- **Project Goals:** The primary goals were to develop a system for efficient drug discovery using molecular deep learning, focusing on drug repurposing and predictive toxicology.
- **Achievements:**
 - **Functional System:** Successfully developed and implemented a functional system that integrates data upload, model training, and result visualization.
 - **User-Friendly Interface:** Designed an intuitive user interface that simplifies the complex process of drug discovery.

- **Accurate Predictions:** Achieved high accuracy in predictive models, providing reliable insights for drug repurposing and toxicity prediction.

4.2 Key Insights and Learnings

- **Technological Insights:** Gained valuable experience in applying deep learning techniques to molecular data and understanding the intricacies of drug discovery processes.
- **User Feedback:** User testing and feedback highlighted the importance of an intuitive interface and real-time performance, leading to iterative improvements.
- **Challenges Overcome:** Addressed challenges related to data preprocessing, model optimization, and system integration, resulting in a robust and scalable solution.

5. Further Development or Research

The final section outlines potential areas for further development or research, providing a roadmap for future enhancements and exploration.

5.1 Additional Features

- **Advanced Visualization Tools:** Implement more sophisticated visualization tools to enhance the interpretation of predictive results.
- **Automated Model Selection:** Develop an automated system for selecting the best-performing models based on specific datasets and criteria.
- **Real-Time Data Processing:** Enable real-time data processing to handle streaming data and provide instantaneous results.

5.2 Improved Model Accuracy

- **Hyperparameter Tuning:** Explore advanced hyperparameter tuning techniques, such as Bayesian optimization or grid search, to further improve model accuracy.
- **Ensemble Methods:** Implement ensemble learning methods to combine predictions from multiple models, enhancing overall predictive performance.
- **Data Augmentation:** Utilize data augmentation techniques to expand the training dataset and improve model robustness.

5.3 Scalability Enhancements

- **Distributed Computing:** Implement distributed computing solutions to handle larger datasets and more complex models, ensuring the system can scale efficiently.
- **Cloud Integration:** Integrate with cloud platforms to leverage their computational power and scalability, providing flexible and scalable solutions for users.

5.4 New Use Cases

- **Personalized Medicine:** Explore the application of molecular deep learning in personalized medicine, tailoring drug treatments to individual genetic profiles.
- **Rare Disease Research:** Investigate the potential of the system in identifying treatments for rare diseases, leveraging deep learning to uncover novel therapeutic opportunities.
- **Environmental Toxicology:** Extend the system's capabilities to assess the environmental impact of chemicals, predicting their toxicity and ecological effects.