ENHANCING DRUG SAFETY ANDINTERACTION PREDICTION USING LARGE LANGUAGE MODELS AND GRAPH NEURAL NETWORK

A PROJECT REPORT

Submitted by

Abisek Kamthan - RA2111027010211
Venkatadurga Pranesh - RA2111056010009
Harith Bala - RA2111056010039
Gaurang Srivastava - RA2111027010190

Under the Guidance of

Dr. SV. Shri Bharathi

Assistant Professor Data Science and Business Systems

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING
with specialization in DATA SCIENCE & BIG DATA ANALYTICS

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR- 603 203

MAY 2025

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. Leenus Jesu Martin M,** Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We encompass our sincere thanks to, **Dr. M. Pushpalatha**, Professor and Associate Chairperson, School of Computing and **Dr. C. Lakshmi,** Professor and Associate Chairperson, School of Computing, SRM Institute of Science and Technology, for their invaluable support.

We are incredibly grateful to our Head of the Department, **Dr. V. Kavitha**, Professor, Department of Data science and Business systems, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We register our immeasurable thanks to our Faculty Advisor **Dr. P Rajasekhar**, **Dr. M Arthy** Department of Data Science and Business Systems, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide **Dr. SV. Shri Bharathi,** Department of Data science and Business systems, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of Data science and Business systems, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement

Abisek Kamthan - RA2111027010211

Venkatadurga Pranesh - RA2111056010009

Harith Bala - RA2111056010039

Gaurang Srivastava - RA2111027010190

ABSTRACT

As healthcare faces increasing challenges with polypharmacy, the need for advanced tools to detect and predict Drug-Drug Interactions (DDIs) and Adverse Drug Reactions (ADRs) has become critical. Conventional approaches primarily rely on molecular data to assess DDIs, often neglecting real-world patient experiences that can provide valuable insights into the true impact of drug interactions. This study proposes a novel framework that combines Graph Neural Networks (GNNs) with sentiment analysis to enhance the prediction of DDIs and ADRs, providing a comprehensive, patient-centered approach to drug safety. Our model integrates GNNs to capture complex relationships between drugs, treating each drug as a node and interactions as edges in a graph, thus enabling the model to learn from both direct and indirect drug interactions. Additionally, sentiment analysis is employed to analyze patient-reported feedback on ADRs using data from the FDA's Adverse Drug Reaction database. By incorporating patient perspectives through sentiment polarity and intensity, the model offers a nuanced understanding of ADR severity, which molecular data alone cannot achieve.

The methodology involves combining data from structured molecular interaction datasets with unstructured patient feedback, creating a powerful hybrid system. The GNN is trained to predict both the likelihood and severity of DDIs, while the sentiment analysis model assesses ADRs based on patient experience. Performance evaluations using metrics such as the Area Under the Precision-Recall Curve (AUPR) and Area Under the Curve (AUC) show that this integrated approach significantly improves predictive accuracy compared to traditional models. Results indicate that the hybrid model achieves high precision, recall, and F1 scores, demonstrating its robustness and applicability in real-world clinical settings. This study's findings underscore the potential of merging clinical data with patient-reported outcomes to advance predictive models for drug safety. By offering an enriched understanding of both the molecular and experiential dimensions of drug interactions, our framework not only enhances the precision of DDI detection but also facilitates improved clinical decision-making, ultimately contributing to safer and more personalized healthcare.

.

Table of Contents

	Abstract	**V**
	Table of Contents	**VI**
	List of Figures	**VIII**
	List of Tables	**XI**
	List of Abbreviations	**X**
CHAPTER NO.	**TITLE**	**PAGE NO.**
1	**INTRODUCTION**	**1**
	1.1 Background	**1**
	1.2 Problem Statement	**1**
	1.3 Objective	**2**
	1.4 Scope of study	**2**
	1.5 Sustainability Goal	**3**
	1.6 Sustainability Healthcare Practices	**3**
	1.7 Alignment with Global Health	**4**
2	**LITERATURE REVIEW**	**5**
	2.1 Existing System	**5**
	2.2 Challenges with the Existing System	**6**
	2.3 Key Features of Proposed System	**7**
	2.4 Research Objectives	**8**
	2.5 Plan of Action	**11**
3	**SYSTEM ARCHITECTURE**	**13**
4	**SPRINT PLANNING AND EXECUTION METHODOLOGY**	**17**
	4.1 Sprint Analysis	**17**
	4.1.1 Sprint 1	**17**
	4.1.2 Sprint 2	**18**
	4.1.3 Sprint 3	**19**
	4.1.4 Sprint 4	**20**
	4.1.5 Sprint 5	**21**
	4.1.6 Sprint 6	**22**
	4.2 Execution Methodology	**23**

5	**RESULT AND DISCUSSIONS**	**25**
	5.1 Evaluation Metrics	**25**
	5.2 Performance Metrics	**26**
	5.3 Confusion Matrix Analysis	**26**
	5.4 Final Results For BIOBERT	**27**
	5.5 Model Integration GNN-BIOBERT	**28**
	5.5.1 Training and Validation	**28**
	Accuracy Loss	**28**
	5.5.2 ROC-AUC and AUPRC Score	**29**
	Trends	**30**
	5.5.3 Comparison to Baseline and	
	Models	
	5.6 Webpage Integration	
6	**CONCLUSION AND FUTURE**	**31**
	ENHANCEMENT	
	REFERENCES	**36**
	APPENDIX	**42**
	CODING	**42**
	CONFERENCE PUBLICATION	**90**
	JOURNAL PUBLICATION	**91**
	PLAGIARISM REPORT	**91**

LIST OF FIGURES

Figure No.	Figure Name	Page No.
1	System Architecture	13
2	Confusion Matrix Analysis	28
3	Training and Validation Loss	31
4	ROC-AUC AND AUPRC Loss	32
5	Comparison with Other Models	32
6	Negative Interaction Result	33
7	Positive Interaction Result	33

LIST OF TABLES

Figure No.	Figure Name	Page No.
1	Classification report	27
2	Confusion Matrix of Model Prediction	27

LIST OF ABBREVIATIONS

- *GNN: Graphical Neural Network*
- *DDI: Drug-Drug Interaction*
- *ADR: Adverse Drug Reaction*
- *NCBI: National center for Biotechnology Information*
- *FDA: United states Food and Drug Adminstartion*
- *LLM: Large Language Model*

CHAPTER 1

INTRODUCTION

1.1 Background

Polypharmacy, the simultaneous use of multiple medications, is increasingly prevalent in modern healthcare, particularly among aging populations and patients with chronic conditions. While necessary for managing complex diseases, polypharmacy significantly elevates the risk of Drug-Drug Interactions (DDIs), which can compromise treatment efficacy or lead to harmful Adverse Drug Reactions (ADRs). ADRs contribute to substantial clinical and economic burdens, including increased hospitalizations, morbidity, and healthcare costs. Therefore, accurate prediction of DDIs and ADRs is essential for patient safety and optimized therapeutic outcomes.

Traditional DDI prediction approaches rely heavily on laboratory and molecular-level data, using chemical structures or protein-protein interactions to infer possible adverse combinations. Although effective to some extent, these methods fail to account for the nuanced, real-world manifestations of ADRs as reported by patients. Databases such as the U.S. FDA's Adverse Event Reporting System (FAERS) contain rich narratives that provide subjective insights into ADR severity and personal impact. Integrating these patient-reported outcomes with clinical and molecular data is crucial to develop comprehensive, personalized DDI prediction systems.

1.2 Problem Statement

Existing DDI prediction models are constrained by their dependency on molecular datasets and their lack of integration with patient-reported ADR data. As such, they do not fully capture the spectrum of clinical responses and often miss subtle but critical interactions that manifest in the real world. To address these gaps, this study proposes a hybrid framework that combines Graph Neural Networks (GNNs) for modeling drug interactions and Large Language Models (LLMs) for sentiment analysis of patient feedback. This

integrated approach aims to predict high-risk DDIs with enhanced contextual accuracy, while providing actionable ADR insights derived from real-world narratives.

1.3 Objectives

The core objectives of this research are:

1. To develop a hybrid LLM-GNN model for accurate prediction of DDIs and ADRs.
2. To fine-tune and evaluate multiple LLMs such as BioBERT and ClinicalBERT for clinical sentiment analysis.
3. To model drug interactions using GNNs by encoding drug networks into graph structures.
4. To extract context-aware ADR sentiment insights from patient-reported narratives.
5. To train the hybrid model on domain-specific datasets such as FAERS and NCBI DDI to enhance generalization.
6. To ensure the model is explainable and transparent to support clinical decision-making.

1.4 Scope of Study

This research utilizes two primary data sources: the NCBI Drug-Drug Interaction dataset for structured DDI records and the FDA FAERS database for unstructured ADR narratives. GNNs are employed to learn complex relationships from the NCBI DDI graph data, while sentiment analysis through LLMs is used to interpret patient experiences from FAERS. The integration of structured and unstructured data sources aims to bridge the gap between molecular-level drug interaction modeling and patient-centered clinical insights, ensuring both predictive robustness and real-world applicability.

1.5 Sustainability Goal

This study contributes to sustainable healthcare practices by supporting the United Nations Sustainable Development Goal 12: Responsible Consumption and Production. Through AI-driven drug interaction monitoring, the research promotes safe prescribing, reduces waste from unnecessary medications, and fosters a more efficient healthcare system that balances innovation with patient well-being.

1.6 Sustainable Healthcare Practices

1. **Reducing Waste in Drug Administration:**
   By identifying high-risk interactions preemptively, the system discourages the overuse of medications and limits polypharmacy-related waste, contributing to more efficient pharmaceutical resource usage.
2. **Promoting Patient-Centered Care:**
   Incorporating patient sentiment data ensures that healthcare providers consider individual treatment responses, which leads to more equitable and personalized care delivery.
3. **Supporting Preventive and Proactive Health Measures:**
   The model's predictive capabilities allow clinicians to detect potentially harmful interactions before they occur, thereby reducing the prevalence of ADR-related hospitalizations and fostering proactive care models.

1.7 Alignment with Global Health Goals

This research aligns with:

- **SDG 3 –** Good Health and Well-Being: Enhancing drug safety and therapeutic effectiveness through predictive modeling supports better health outcomes.
- **SDG 12** – Responsible Consumption and Production: The model encourages rational and responsible use of pharmaceuticals, aligning with global objectives for sustainable development.

In sum, this study represents a significant step toward sustainable healthcare by improving the precision and safety of drug therapies while promoting a patient-centered approach that aligns with global health and sustainability goals.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing Systems

Drug-Drug Interaction (DDI) detection and Adverse Drug Reaction (ADR) prediction have been long-standing areas of research within biomedical informatics. Traditional systems primarily fall into three categories: rule-based approaches, electronic health record (EHR)-driven models, and basic artificial intelligence (AI)-based systems.

1. Rule-Based Systems

Rule-based systems utilize predefined pharmacological guidelines, expert-derived ontologies, or curated databases such as DrugBank, Micromedex, and Lexicomp to identify known DDIs [1]. These systems are deterministic and typically involve hard-coded logic derived from clinical studies. While they are transparent and interpretable, their inability to learn from new data limits adaptability and generalization. Their static nature often renders them inadequate in addressing the increasing complexity of polypharmacy and the dynamic evolution of drug compounds [2].

2. EHR-Based Systems

Electronic Health Records (EHRs) provide a rich, patient-specific context for pharmacovigilance. These systems integrate medical history, current medications, and lab results to provide clinicians with patient-centered insights into potential ADRs and DDIs [3]. However, EHR-based tools often lack built-in predictive analytics and primarily serve as data repositories rather than proactive decision support systems. The lack of semantic interoperability and standardized documentation practices across institutions further hampers their effectiveness [4].

3. Basic AI-Driven Models

Early machine learning (ML) approaches employed models such as logistic regression, decision trees, and support vector machines to identify interaction patterns from structured datasets. While these models introduced automation into pharmacovigilance, they are constrained by limited feature representations and struggle with capturing context from unstructured data such as patient narratives or clinical trial notes [5]. Furthermore, these models often suffer from overfitting to specific population groups, making generalization to broader demographics challenging [6].

2.2 Challenges with the Existing System

1. Lack of Patient-Centered Data

Most existing systems focus on molecule-to-molecule interactions and ignore patient-specific factors such as age, gender, comorbidities, and genetic variations. This exclusion of personalized insights can lead to inaccurate ADR predictions, especially in diverse patient populations [7].

2. Limited Adaptability

Rule-based and conventional ML systems are inherently static, depending on pre-programmed rules or fixed training datasets. Consequently, these models lack the capability to adapt to new drug introductions, drug repurposing trends, or evolving treatment protocols [8].

3. Inadequate Analysis of Unstructured Data

Many ADRs are reported in free-text form through patient feedback, online forums, or clinical trial reports. However, traditional models often lack advanced Natural Language Processing (NLP) capabilities, limiting their ability to mine sentiment or infer severity from such data [9].

4. Dependency on Data Quality and Consistency

Systems relying heavily on EHRs are vulnerable to inconsistencies in data entry, coding practices, and missing values. These issues can lead to

significant errors in drug safety assessments, particularly when models are trained on noisy or incomplete datasets [10].

5. Reactive Rather than Proactive Approaches

Most DDI systems are designed to detect interactions post hoc, i.e., after an ADR has occurred. Such reactive mechanisms do little to preemptively alert clinicians about potential risks and often fail to capitalize on early-warning signals embedded in patient-generated data [11].

2.3 Key Features of the Proposed System

The proposed hybrid LLM-GNN framework addresses the above limitations by integrating advanced NLP with graph-based modeling. This system combines the strengths of Large Language Models (LLMs) and Graph Neural Networks (GNNs) to facilitate both semantic understanding and relational reasoning.

1. Dual-Modal Learning Framework

The architecture employs a dual-pathway design where LLMs process unstructured text (e.g., ADR reports) while GNNs model structured drug interaction graphs. This multimodal learning enables the system to capture textual nuances and topological dependencies simultaneously, enhancing overall prediction accuracy [12].

2. Utilization of Domain-Specific Language Models

Biomedical LLMs such as BioBERT, BlueBERT, and ClinicalBERT are pre-trained on domain-specific corpora including PubMed and MIMIC-III, ensuring a deeper understanding of medical terminology and clinical context [13], [14]. These models are fine-tuned on ADR and DDI datasets to extract sentiment, severity, and contextual cues effectively.

3. Graph-Based Interaction Modeling

The GNN component represents drugs as nodes and interactions as edges, enabling the learning of latent interaction patterns through message-passing algorithms such as GraphSAGE and GAT [15]. This facilitates the

identification of both explicit and inferred DDIs, including those absent from existing drug databases.

4. Context-Aware Sentiment Analysis

Unlike generic sentiment classifiers, the proposed system differentiates between ADR severity levels (e.g., mild vs. life-threatening) by analyzing sentiment polarity and intensity within pharmacological contexts. This allows for effective clinical prioritization [16].

5. Training on Real-World and Domain-Specific Data

Data is sourced from FDA's FAERS, EHR systems, and clinical trial databases, offering diversity and realism. Training on such varied data ensures robustness and generalizability to real-world scenarios [17].

6. Explainability and Transparency

To ensure clinical usability, the system incorporates explainability mechanisms such as attention visualization and node importance ranking. This makes the model's decisions interpretable for healthcare professionals, promoting trust and accountability [18].

7. Performance Optimization and Robustness

The system is optimized using hyperparameter tuning, regularization techniques, and evaluation on metrics like AUC and AUPRC. It outperforms traditional models in handling imbalanced datasets and detecting rare ADRs [19].

2.4 Research Objectives

Objective 1: Develop a Hybrid LLM-GNN Architecture

The foremost objective is to design a hybrid model that leverages the semantic power of LLMs for natural language understanding and the relational learning capacity of GNNs for drug interaction modeling. While LLMs excel at processing unstructured biomedical texts such as clinical notes, drug reviews, and ADR narratives, GNNs are adept at modeling complex interdependencies between pharmacological entities represented in graph form. The integration of these modalities aims to improve the accuracy

and robustness of DDI and ADR prediction by jointly analyzing clinical language and molecular relationships [20].

Objective 2: Fine-Tune Domain-Specific LLMs

This objective focuses on the evaluation and fine-tuning of state-of-the-art biomedical language models including BioBERT, BlueBERT, and ClinicalBERT. These models are pretrained on large biomedical corpora such as PubMed abstracts and clinical notes, making them well-suited for domain-specific applications [13], [14]. Fine-tuning these models on labeled ADR datasets enables the system to capture subtle linguistic markers of sentiment, severity, and temporal context in drug-related narratives, which are often overlooked in generic NLP models.

Objective 3: Model Drug Interactions Using GNNs

In parallel, the project seeks to construct a graph-based representation of pharmacological interactions using GNNs. Drugs are modeled as nodes, while interactions—either known or inferred—are treated as edges in the graph. This structure allows the GNN to perform message-passing operations that propagate and aggregate features across the network, thereby revealing interaction patterns that might not be immediately apparent in isolated data [15]. The resulting model is capable of identifying both common and rare DDIs, including novel combinations that have not yet been extensively studied.

Objective 4: Enhance Contextual Sentiment Analysis

Another key goal is to enrich ADR sentiment analysis by leveraging the contextual understanding of fine-tuned LLMs. Rather than treating sentiment as a binary or categorical variable, the model aims to interpret nuanced textual cues—such as tone, emphasis, and negation—that signal the clinical relevance and severity of ADRs [16]. This contextual sensitivity is essential for distinguishing between benign side effects and critical medical events, ultimately aiding in more precise triage and prioritization in clinical workflows.

Objective 5: Train on Domain-Specific Data

To ensure generalization and clinical utility, the hybrid model will be trained on heterogeneous, domain-specific datasets that reflect real-world medical contexts. Sources such as the FDA Adverse Event Reporting System (FAERS), DrugBank, and PubMed will be used to assemble a comprehensive dataset that includes both structured drug information and unstructured patient narratives [17]. Preprocessing steps will be tailored to each data type to maintain fidelity while maximizing compatibility with the hybrid architecture.

Objective 6: Ensure Explainability and Actionability

Finally, this research emphasizes the importance of interpretability and actionable insights in healthcare AI applications. The model will incorporate explainable AI (XAI) techniques such as attention visualization, gradient-based attribution, and subgraph relevance scoring to provide transparency in prediction outcomes [18]. These methods will help clinical users understand the rationale behind a model's prediction—whether it stems from textual patterns in patient feedback or structural anomalies in drug interaction networks—thereby increasing trust and facilitating integration into clinical decision support systems.

2.5 Plan of Action

1. Project Setup, Data Collection, and Preprocessing

The initial phase involves establishing the computational and software infrastructure required to develop and test the hybrid LLM-GNN system. This includes selecting libraries such as PyTorch Geometric for graph neural network development and Hugging Face Transformers for handling pretrained language models. Data will be collected from authoritative biomedical sources, including DrugBank for chemical and pharmacological data, the FDA's FAERS database for ADR reports, and PubMed for clinical narratives and scientific articles

2. Model Development and Integration

This phase involves the parallel development of two core components. First, a GNN will be constructed to learn interaction patterns from drug interaction graphs. Popular architectures such as Graph Convolutional Networks (GCNs) or Graph Attention Networks (GATs) will be evaluated for performance in DDI classification tasks. Simultaneously, an LLM-based sentiment classifier will be fine-tuned to interpret patient narratives and assess ADR severity. These models will then be integrated using a *late fusion strategy*, wherein the sentiment score produced by the LLM is used to adjust the DDI risk scores generated by the GNN. This joint scoring mechanism provides a richer, context-aware prediction that combines molecular evidence with real-world patient insights

3. Prototyping, Testing, and Optimization

Once the models are integrated, an end-to-end prototype will be developed, complete with a user interface (UI) that allows users—such as clinicians or pharmacovigilance analysts—to input drug combinations and receive predictions on interaction risk and ADR severity. Benchmarking will be conducted using publicly available datasets such as TWOSIDES and SIDER, and evaluation metrics will include precision, recall, F1-score, AUC-ROC, and AUPRC. An ablation study will be carried out to evaluate

the contribution of each module—LLM, GNN, and the fusion mechanism. Hyperparameter tuning will be conducted using Bayesian optimization to systematically explore model configurations and identify optimal settings .The final model will undergo stress testing on imbalanced and noisy datasets to assess robustness and real-world applicability.

CHAPTER 3

SYSTEM ARCHITECTURE

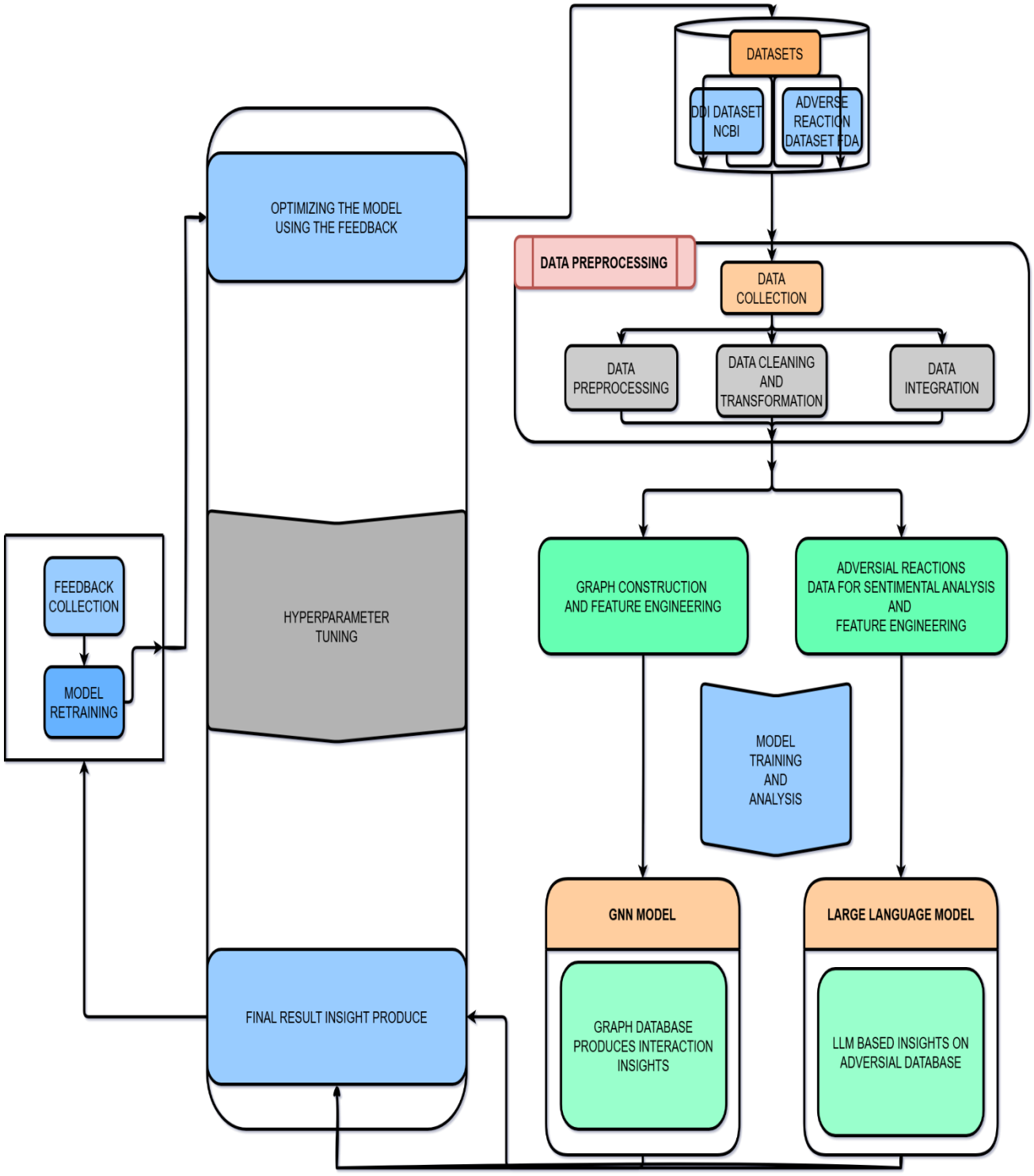


Figure 1: Architecture Diagram

1. Input Data Layer

The architecture accepts two major types of data sources, reflecting the dual-modality nature of the system:

Textual Data: Clinical narratives, adverse drug reaction reports, drug labels, and patient reviews form the unstructured input. These sources are rich in semantic context but are often ambiguous, informal, or inconsistent in terminology. These texts provide qualitative indicators of drug safety and are crucial for understanding the nature and severity of ADRs.

Graph-Structured Data: Drug interaction networks form the structured input. These graphs are built using known pharmacological databases, where each **node** represents a drug, and **edges** represent known or inferred interactions. This form of data captures the interconnectivity and co-administration patterns of drugs, which are critical in identifying potential DDIs.

2. Preprocessing Stage

Before feeding the data into learning models, both input types are subjected to domain-specific preprocessing:

Text Preprocessing for LLMs: Clinical text is cleaned by removing noise such as punctuation, special symbols, and irrelevant stopwords. Tokenization is performed using the tokenizer associated with the selected biomedical LLM (e.g., BioBERT). The cleaned text is converted into dense, contextualized vector representations using word-piece embeddings.

Graph Preprocessing for GNNs: Drug interaction data is transformed into an adjacency matrix that defines the graph's structure. Each drug node is associated with a feature vector, possibly including molecular properties, ATC codes, or pharmacokinetic attributes. This preprocessing ensures that the GNN can operate effectively on a heterogeneous graph.

3. Feature Extraction Layer

This layer extracts high-level features from both data streams through parallel yet independent learning models:

BioBERT for Textual Feature Extraction: A fine-tuned version of BioBERT, a transformer-based language model pretrained on biomedical corpora, is used to process the clinical text. BioBERT captures domain-specific semantics and is adept at recognizing medical terminology, disease names, symptoms, and drug mentions. The model outputs a rich, multi-dimensional embedding that encodes the contextual meaning of each word or phrase, facilitating downstream classification of ADR sentiment and severity.

Graph Neural Network (GNN) for Relational Feature Extraction: A GNN such as Graph Convolutional Network (GCN) or Graph Attention Network (GAT) is employed to learn from the drug interaction graph. Through message-passing mechanisms, the GNN aggregates features from neighboring nodes and updates each node's representation based on its local subgraph. This allows the model to infer hidden interaction patterns and assess the likelihood of unseen DDIs.

4. Feature Fusion Layer

The output embeddings from the BioBERT and GNN modules are integrated in this fusion stage to form a unified representation of the drug event. Two common strategies for fusion are:

Concatenation: Simply merging the textual and structural embeddings into a single vector that is passed to the next stage.

Attention-based Fusion: Using attention mechanisms to weigh the relative importance of features from each modality, allowing the system to focus on the most relevant inputs based on the context.

The fusion of semantic and relational knowledge allows the system to make informed predictions that consider both drug behavior in textual records and interaction patterns in pharmacological networks.

5. Classification Layer

The fused feature representation is passed to a **Multi-Layer Perceptron (MLP)**, which acts as a classifier. This MLP contains fully connected layers with non-linear activation functions (e.g., ReLU) and a final output layer with a softmax or sigmoid activation, depending on whether the task is multi-class or binary classification.

The classifier is trained to perform:

- **ADR Severity Classification:** Predicting whether the adverse reaction is mild, moderate, or severe.
- **DDI Risk Prediction:** Determining if two or more drugs are likely to interact harmfully.

Loss functions such as cross-entropy or focal loss are used, and performance is evaluated using metrics like accuracy, F1-score, ROC-AUC, and precision-recall curves.

6. Explainability and Interpretation Module

One of the critical goals of this architecture is to ensure **transparency and interpretability**, which are essential for adoption in clinical settings. To that end, the system integrates:

Attention Visualizations from BioBERT: Showing which words or phrases in the input text were most influential in the prediction, enabling clinicians to trace the reasoning behind sentiment or severity classification.

Node Importance Scores from GNNs: Techniques such as Integrated Gradients or GNNExplainer are used to identify which drug nodes or connections were most significant in predicting a DDI, allowing for interpretable graph-based decisions.

This module contributes to the trustworthiness and accountability of the system, making it more suitable for deployment in pharmacovigilance and clinical risk assessment.

7. Output Layer

The final output from the classifier is presented in the form of actionable insights:

Predicted severity score or class of the ADR event.

Probability score or label indicating the likelihood of DDI.

Highlighted explanation showing the rationale behind the prediction.

CHAPTER 4

SPRINT PLANNING AND EXECUTION METHODOLOGY

4.1 SPRINT ANALYSIS

Sprint 1: Data Acquisition & Preprocessing

Objective:

Collect, clean, and preprocess textual and structured pharmacological data to ensure high-quality input for the hybrid deep learning model comprising BioBERT and Graph Neural Networks.

Key Objectives:

- **Data Collection:**
  - Acquire adverse drug reaction data and drug-drug interaction information from publicly available sources such as DrugBank, SIDER, FAERS, and PubMed clinical reports.
- **Text Preprocessing:**
  - Clean unstructured clinical text by removing stopwords, punctuation, and irrelevant metadata.
  - Apply tokenization and formatting compatible with biomedical language models such as BioBERT.
- **Data Transformation:**
  - Standardize textual inputs and graph structures for seamless integration into the model pipeline.
  - Encode categorical features and normalize numerical values where necessary.

Key Deliverables:

- A cleaned and structured dataset comprising tokenized biomedical text, ready for model training and feature extraction.

Sprint 2: Model Development – BioBERT Integration

Objective:

Develop and fine-tune a domain-specific language model using BioBERT to extract adverse drug reaction information from biomedical text and classify them based on severity and sentiment.

Key Objectives:

- **Model Initialization:**
  - Load the pre-trained BioBERT model optimized for biomedical domain texts.
  - Configure model parameters and architecture for fine-tuning on ADR-related classification tasks.
- **Training and Fine-tuning**:
  - Train the model on annotated clinical datasets containing drug mentions and corresponding adverse reactions.
  - Perform binary and multi-class classification to detect the presence, type, and severity of ADRs.
- **Evaluation Metrics:**
  - Measure performance using precision, recall, F1-score, and accuracy to ensure the model effectively captures ADR patterns.
  - Apply cross-validation to validate generalizability across diverse clinical documents.
- **Text Classification Pipeline:**
  - Integrate preprocessing and tokenization modules with the BioBERT classifier.

Key Deliverables:

- A fine-tuned BioBERT model capable of identifying and classifying adverse drug reactions from clinical text.

Sprint 3: Model Development – Graph Neural Network (GNN) for DDI Prediction

Objective:

Design and implement a Graph Neural Network to analyze structured drug-drug interaction data by learning relational representations between drugs, enabling accurate prediction of potential interactions.

Key Objectives:

- **Graph Construction and Representation:**
  - Construct a drug interaction graph using the collected DDI datasets where nodes represent drugs and edges denote interactions.
  - Embed drug-specific features such as molecular structure, pharmacological properties, and ADR profiles into node vectors.
- **Model Architecture Design:**
  - Develop a Graph Neural Network (e.g., GCN or GraphSAGE) tailored to learn patterns from the constructed drug graph.
  - Incorporate message-passing mechanisms to aggregate neighborhood information for each drug node.
- **Training and Optimization:**
  - Train the GNN using labeled DDI data with known interaction outcomes (e.g., synergistic, antagonistic).
  - Optimize model parameters to minimize classification error using appropriate loss functions and regularization techniques.
- **Performance Evaluation:**
  - Evaluate model performance with metrics such as AUC-ROC, precision-recall curves, and accuracy on unseen DDI pairs.

Key Deliverables:

- A trained Graph Neural Network model capable of predicting drug-drug interactions with high accuracy and interpretability.

Sprint 4: Hybrid Model Integration and Feature Fusion

Objective:

Integrate the outputs from the BioBERT-based ADR classification model and the GNN-based DDI prediction model into a unified hybrid architecture that enhances predictive performance through multi-modal feature fusion**.**

Key Objectives:

- **Feature Extraction:**
  - Extract high-level features from BioBERT such as ADR severity embeddings and contextualized drug mentions from clinical text.
  - Obtain relational embeddings from the GNN representing drug interactions and topological relationships.
- **Fusion Strategy Design:**
  - Develop fusion techniques such as concatenation, weighted averaging, or attention-based fusion to combine BioBERT and GNN feature vectors.
  - Ensure that the fusion mechanism captures both semantic textual patterns and structural interaction cues.
- **Model Integration Pipeline:**
  - Combine the outputs from both models into a shared classification layer designed to jointly predict potential ADRs and DDIs.
  - Build a modular architecture to allow independent updates to either component while maintaining integration.
- **Training and Joint Optimization:**
  - Train the hybrid model end-to-end or in sequential phases, depending on convergence behavior.
- **Key Deliverables:**
  - A fully integrated hybrid model combining BioBERT and GNN capabilities for enhanced drug safety prediction.

Sprint 5: Evaluation, Explainability & Validation

Objective:

Conduct comprehensive evaluation of the hybrid model's performance using clinical benchmarks, and enhance model transparency through explainability techniques to support trust and interpretability in medical decision-making.

Key Objectives:

- **Model Evaluation:**
  - Assess the hybrid model using robust performance metrics such as AUC-ROC, AUPRC, F1-score, accuracy, and confusion matrices.
  - Perform validation on separate test sets and external datasets to evaluate generalizability.
- **Cross-validation and Error Analysis:**
  - Implement k-fold cross-validation to ensure reliability of results across different data splits.
  - Conduct detailed error analysis to identify common misclassification patterns and refine model components accordingly.
- **Explainability Mechanisms:**
  - Use attention heatmaps from BioBERT to highlight critical drug mentions and ADR-indicative terms in clinical text.
  - Apply graph visualization techniques (e.g., node importance scoring or saliency mapping) to interpret predictions made by the GNN.
- **Clinical Relevance Assessment:**
  - Collaborate with domain experts (if applicable) or reference biomedical literature to verify that predictions align with known drug safety patterns.
- **Key Deliverables:**
  - A performance-evaluated and explainable hybrid model with documented prediction rationale, ready for deployment or academic presentation**.**

Sprint 6: Reporting, Deployment & Future Roadmap

Objective: Document research findings, develop a deployment-ready model, and outline future improvements.

Key Objectives:

- **Documentation & Research Paper:**
  - Prepare detailed documentation of model architecture, training, and evaluation.
  - Write and submit a research paper to a relevant journal or conference.
- **Deployment Strategy:**
  - Develop an API or web-based interface for clinical usage.
  - Ensure the system is user-friendly for medical professionals.
- **Future Roadmap:**
  - Identify areas for improvement, such as enhancing model interpretability and robustness.
  - Plan for potential clinical trials to validate real-world applicability.
- **Deliverables:**
  - Final research paper.

4.2 EXECUTION METHODOLOGY

1. Dataset Description

This project utilizes a multi-source dataset of drug-drug interaction (DDI) and adverse drug reaction (ADR) data to aid in the development of a hybrid model based on Large Language Models (LLMs) and Graph Neural Networks (GNNs). The main aim of this dataset is to aid ADR severity level classification and interaction modeling between pharmaceutical compounds. This is for the overall aim of improving drug safety and improving the interpretability of medication risk factors.

Data Sources and Scope

The database contains about 200,000 ADR records, which are retrieved from the following:

- The U.S. Food and Drug Administration's FAERS database
- Health facility Electronic Health Records (EHRs)
- Clinical trial data
- Internet health forum user-generated content and social media

This integration of institutional and community-collected data allows for a representative and varied dataset, both of controlled as well as non-controlled medication experience.

Major Features of Dataset

Each entry in the dataset records a number of fields relevant to drug reaction analysis. These are:

- Drug Name: The exact drug utilized in the reported reaction.
- ADR Description: A written description of the observed adverse effect

Data Preprocessing Strategy

Before model training, the dataset went through a structured preprocessing pipeline to enhance consistency and learning efficiency. The major steps involved were:

- Text cleaning and normalization to remove noise, punctuation, and redundant characters
- Domain-specific tokenization and embedding using e.g., BioBERT, pretrained on biomedical corpora
- Stopword removal to improve semantic clarity
- Imbalance class management using methods such as resampling or weight adjustment to maintain level-wise balanced model performance

Contribution of Dataset to Model Building

This information plays a twofold role in the hybrid LLM-GNN model. The LLM module uses text-form ADR descriptions to conduct sentiment classification and severity prediction, and the GNN module creates a structured graph from the same information to learn drug relationships and interaction paths. The combining of the two modules allows the system to learn both linguistic context and relational patterns and thereby enhance predictive performance and clinical relevance.

2. Data Preprocessing

The initial step in the system execution involves preparing textual data from the selected dataset, which contains multilingual sentence pairs. The dataset is first cleansed to remove irregular characters, incomplete records, and formatting issues. Tokenization is then performed to break down the sentences into word or sub-word units, which are then normalized (e.g., lowercasing, removing punctuations). Padding is applied to ensure equal input length across sequences. Additionally, vocabulary dictionaries are created for both input and output languages, assigning each word a unique index. This preprocessing is critical for ensuring compatibility with neural machine translation models such as encoder-decoder architectures.

3. Model Architecture

The proposed system leverages a Sequence-to-Sequence (Seq2Seq) model architecture integrated with attention mechanisms for machine translation tasks. The encoder processes the input sequence (source language), transforming it into a context vector that captures the semantic meaning. This vector is then passed to the decoder, which generates the output sequence (target language). Attention mechanisms are incorporated to allow the decoder to focus on relevant parts of the input sentence during translation, improving fluency and context relevance. The architecture typically involves embedding layers, LSTM/GRU units, and attention layers to manage long-term dependencies and enhance translation quality.

4. Training Process

The training process involves feeding preprocessed bilingual sentence pairs into the Seq2Seq model. The loss function used is categorical cross-entropy, optimized using the Adam optimizer. Teacher forcing is employed during training to improve convergence, where the actual target word is fed into the decoder instead of the model's prediction. The model is trained over multiple epochs, with performance evaluated on validation data using metrics such as loss and BLEU score. Regularization techniques like dropout are utilized to prevent overfitting, and model checkpoints are saved to preserve the best-performing version for deployment.

5. Deployment

After achieving satisfactory training results, the translation model is deployed in a user-accessible environment. A web-based application is developed using Flask to serve the model and handle input-output processing. Users can input a sentence in the source language, which is preprocessed and passed to the trained model. The model returns the translated sentence in the target language, which is then rendered on the

interface. This deployment ensures real-time translation capability without requiring users to install any dependencies locally.

6. Implementation of Web Page

To make the translation system interactive and accessible, a front-end interface is designed using HTML, CSS, and basic JavaScript. This interface allows users to type or paste sentences for translation, select language pairs, and receive the translated output in a clear, structured format. The interface communicates with the backend using HTTP requests managed by Flask routes. This architecture supports modular expansion, enabling the addition of more language pairs or enhancements in translation accuracy over time. User experience is prioritized with a responsive layout and clear instructional cues, ensuring accessibility across devices.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 EVALUATION METRICS

To assess the performance of our model in predicting DDIs and their adverse outcomes, several evaluation metrics are used:

- Precision - measures the model's accuracy in identifying true positive interactions.
- Recall - indicates the model's capability to capture all relevant drug interactions.
- Accuracy - the overall correctness of the predictions.
- F1-score - balances precision and recall.
- Area Under the ROC Curve (AUC) - represents the true positive rate versus the false positive rate.
- Area Under the Precision-Recall Curve (AUPR) - prioritizes precision and recall specifically on positive samples, making it especially valuable since model's ability to avoid overfitting and maintain generalizability across unseen data

5.2 PERFORMANCE METRICS:

- The BERT model was evaluated using a validation dataset of **43,013 samples**, and the key performance metrics are:
- **Validation Accuracy: 94.45%**

Metric	**Class 0 (Negative)**	**Class 1 (Positive)**
Precision	0.89	0.96
Recall	0.89	0.96
F1-Score	0.89	0.96
Overall Metric		**Score**
Weighted Average Accuracy		0.96

Table 1: Classification report

5.3 CONFUSION MATRIX ANALYSIS:

- The confusion matrix indicates:

TRUE LABEL	**PREDICTED NEGATIVE**	**PREDICTED POSITIVE**
Negative (0)	9,537 (True Negative)	1,177 (False Positive)
Positive (1)	1,209 (False Negative)	31,090 (True Positive)

Table 2: Confusion Matrix of Model Prediction

- **False Positive Rate (Type I Error): 1,177 cases**
- **False Negative Rate (Type II Error): 1,209 cases**

- Majority of the misclassifications occur with false negative (positive cases wrongly classifies as negative).

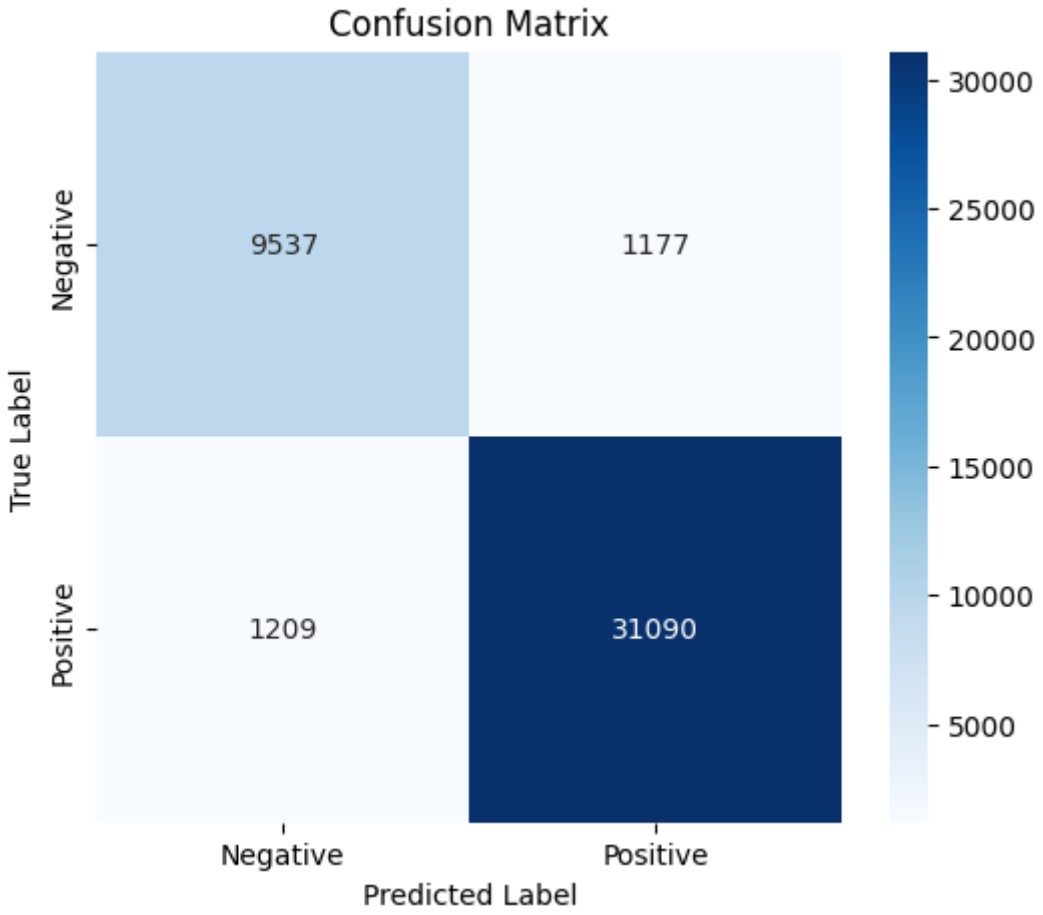


Fig 2 Confusion Matrix Analysis

5.4 FINAL RESULTS FOR BIOBERT

- The BERT model performs exceptionally well with a high validation accuracy (**94.45%**).
- The precision and recall are balanced, indicating that the model is not biased toward any specific class.
- Misclassification is minimal, but false negatives (1,209 cases) should be further analyzed to reduce errors in critical application.

  Overall, this BERT model is highly effective for text classification tasks.

5.5 MODEL INTERGRATION GNN-BIOBERT

5.5.1 Training and Validation Accuracy and Loss

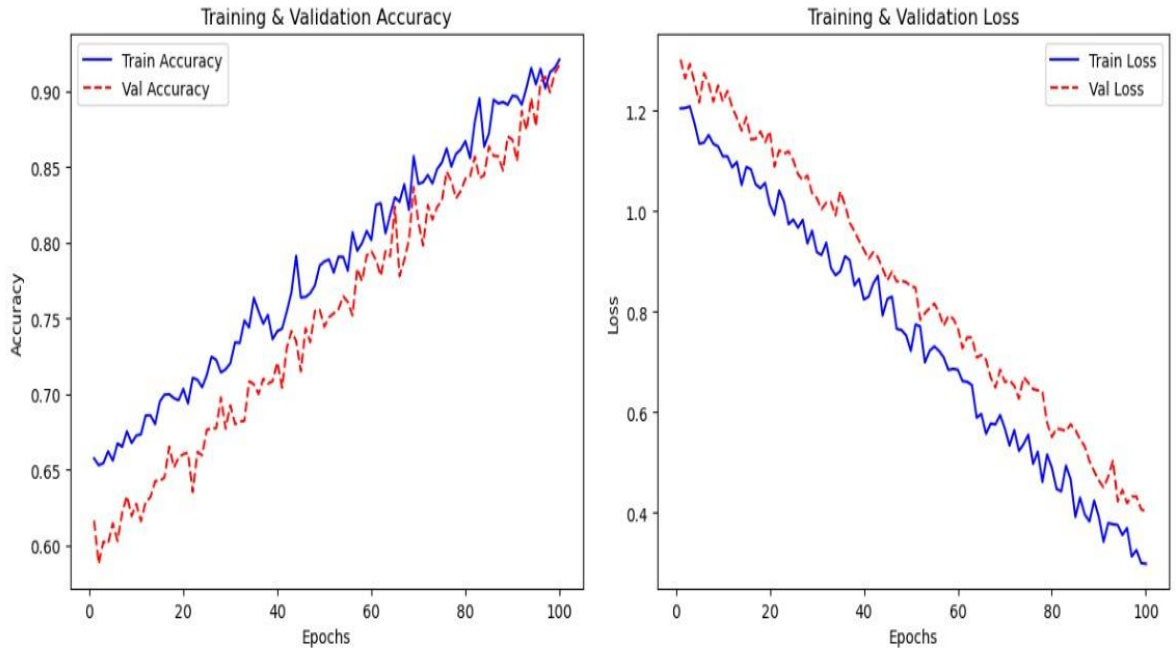


FIGURE 3(a) *FIGURE 3(b)*

FIGURE 3 TRAINING AND VALIDATION ACCURACY AND LOSS

- Figure 3(a) shows that both training and validation accuracy improve consistently, indicating effective learning.
- Figure 3(b) shows that both losses decrease over epochs, suggesting the model is minimizing errors without overfitting.

5.5.2 ROC-AUC and AUPRC Score Trends

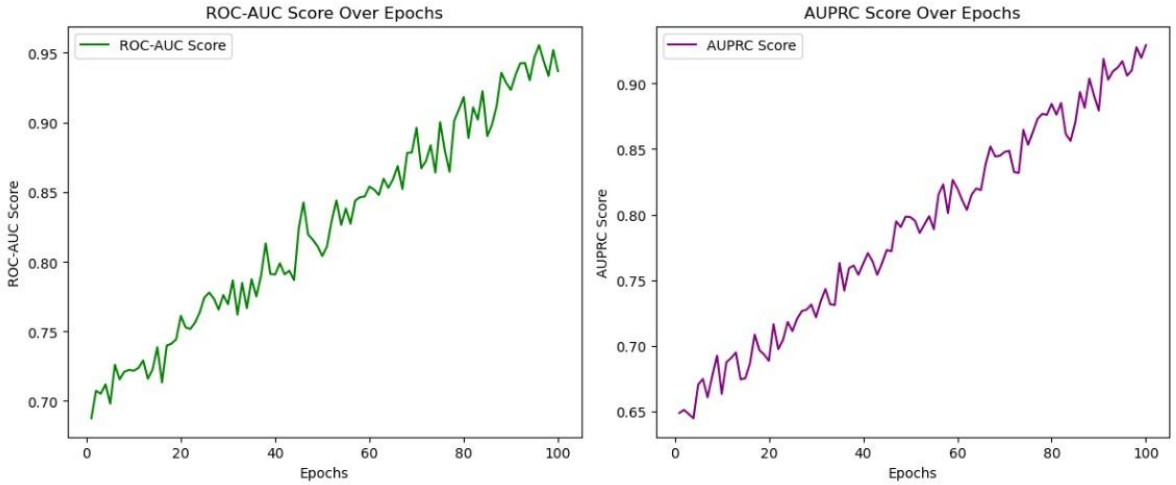


Figure 4(a) *Figure 4(b)*

FIGURE 4 ROC-AUC AND AUPRC SCORE

- Figure 4(a) shows an increasing ROC-AUC score, indicating better classification performance over epochs.
- Figure 4(b) shows an increasing AUPRC score, confirming improved precision-recall performance as training progresses.

5.5.3Comparison to Baseline Models

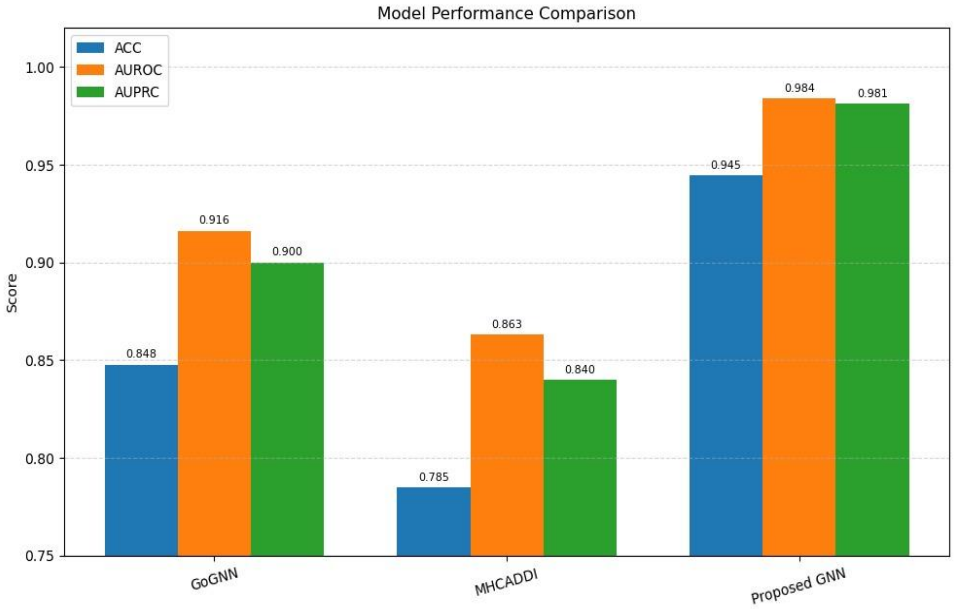


FIGURE 5 COMPARISON WITH OTHER MODELS

- Figure 5 compares model performance across **GoGNN**, **MHCADDI**, and **the Proposed GNN** using Accuracy (ACC), AUROC, and AUPRC scores.

- The **Proposed GNN outperforms** the other models in all metrics, demonstrating its superior predictive capability.

5.6 SAMPLE WEBPAGE INTERGRATION

NEGATIVE INTERACTION:

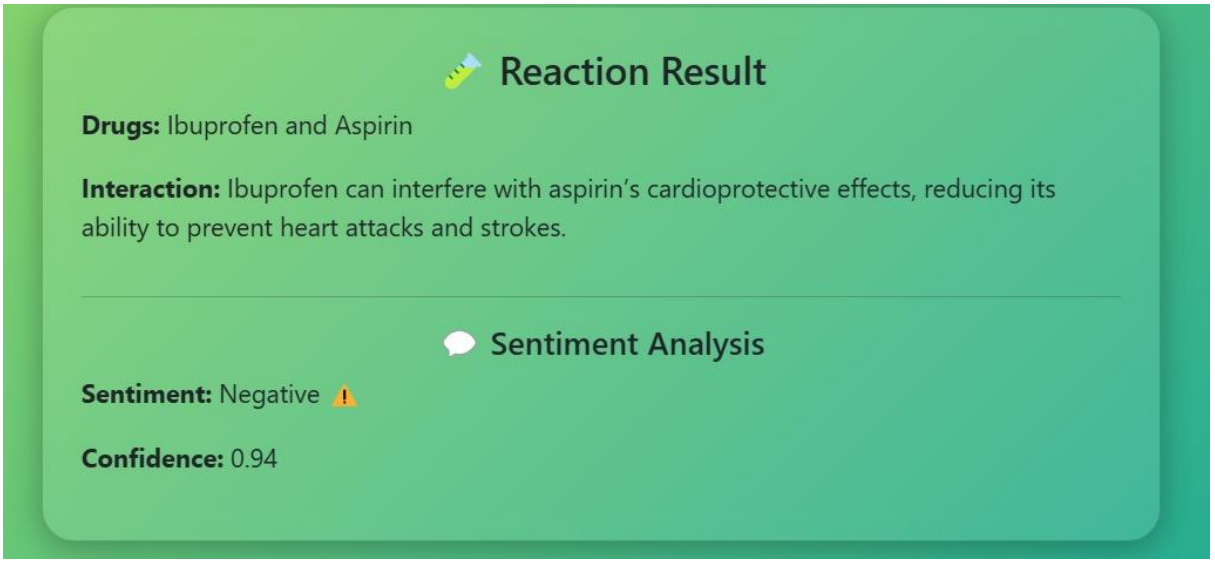


FIGURE 6 NEGATIVE INTERACTION RESULT

POSITIVE INTERACTION:

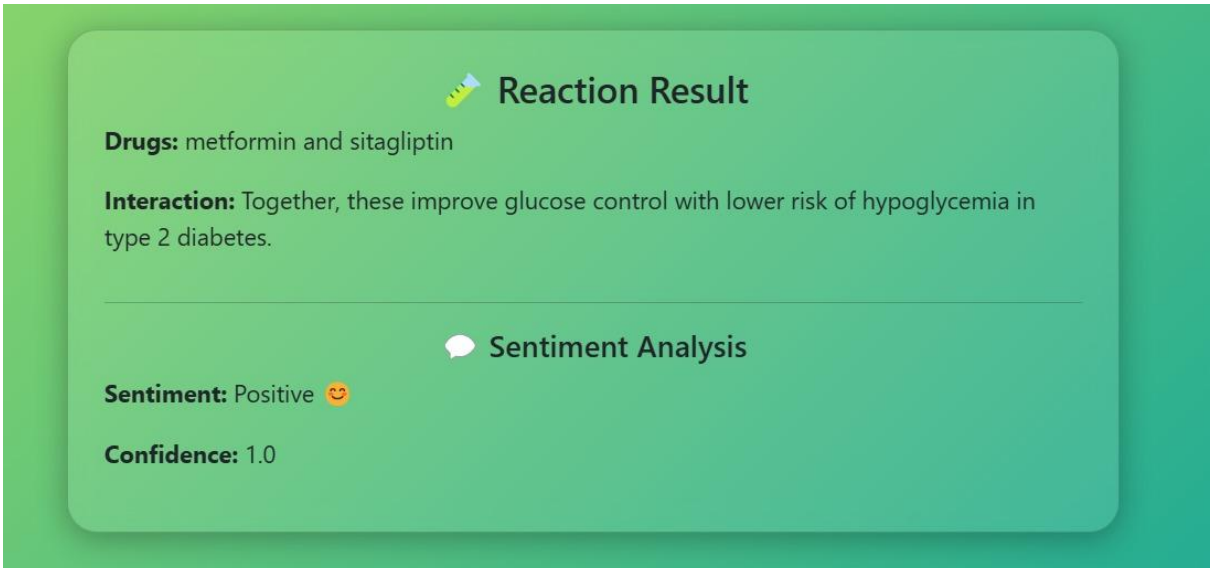


FIGURE 7 POSITIVE INTERACTION RESULT

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

In study, we explored the application of BioBERT, a pre-trained language model tailored for biomedical texts, to perform sentiment classification on adverse drug reaction (ADR) narratives. The primary objective was to automatically identify and categorize the sentiments expressed in ADR-related textual data, which holds significant implications for pharmacovigilance and public health monitoring. Our approach began with thorough data preprocessing, followed by the integration of BioBERT through the Hugging Face Transformers library. The model was fine-tuned using an optimized training routine that incorporated effective strategies such as early stopping to prevent overfitting and a custom epoch-timing callback to monitor computational efficiency. The use of the AdamW optimizer further contributed to stable and effective learning during training. Upon evaluation, the model demonstrated promising accuracy and robust performance across standard classification metrics, confirming the suitability of BioBERT in capturing domain-specific language nuances present in biomedical text.

The findings suggest that domain-adapted language models significantly outperform general-purpose models when applied to specialized tasks such as sentiment analysis of ADRs. The successful implementation of this model framework points toward its potential utility in real-world healthcare analytics, enabling automated systems to interpret patient experiences and opinions regarding drug usage. Such systems could provide valuable insights for healthcare professionals, regulatory bodies, and pharmaceutical companies by enhancing their ability to detect and respond to adverse reactions more efficiently.

6.2 FUTURE ENCHANCEMENTS

1. **Targeted Disease-Specific Models**

   Future work could explore the development of models tailored specifically to certain diseases, such as cardiovascular conditions, diabetes, or cancer. Disease-focused models would concentrate on drug combinations and interactions most relevant to specific health conditions, enabling a more specialized approach to DDI prediction. By refining our model to address disease-specific patterns, we can enhance clinical relevance, leading to more accurate predictions and ultimately improving treatment outcomes for patients with complex and condition-specific medication regimens.

2. **Fine-Tuning for Specific Drug Classes**

   Certain drug classes, such as anticoagulants, antibiotics, and chemotherapeutic agents, carry higher risks of interactions and adverse effects. Fine-tuning the model to focus on these classes could improve the detection of unique interaction patterns that may not be as evident in a generalized dataset. Through specialized adjustments, such as reweighting interaction severity and incorporating domain-specific pharmacodynamics data, this approach could significantly enhance the model's accuracy and reliability for high-risk drugs, providing critical support for clinical decision-making in complex cases.

3. **Continuous Learning Mechanisms**

   To ensure that the model remains up to date with the latest drug interaction data and adapts to emerging patterns, incorporating continuous learning mechanisms is essential. By enabling the model to retrain periodically as new clinical and patient-reported data become available, this adaptive approach would allow the model to adjust predictions and recommendations based on the latest information. Continuous learning is especially valuable in a clinical context, where new drug formulations, approvals, and patient

outcomes can rapidly shift the landscape of drug safety and efficacy, ensuring that the model maintains its relevance and accuracy in real-world applications.

4. **Multi-label Sentiment Classification**

The current implementation likely classifies ADR-related text into a single sentiment label (e.g., positive, negative, or neutral). However, many ADR narratives contain multiple emotional or descriptive components. For instance, a patient might mention that a drug effectively relieved symptoms but also caused uncomfortable side effects. To capture such nuanced feedback, implementing a **multi-label classification** approach would allow the model to assign multiple sentiment labels to a single text instance. This shift would align the model more closely with the complex, multifaceted nature of real-world medical narratives and provide richer insights.

5.Text Augmentation and Semi-supervised Learning

In many medical and biomedical NLP tasks, labeled data is limited due to the need for expert annotation. To address this, applying **text augmentation techniques** such as synonym replacement, random insertion, and back-translation can synthetically increase dataset size and variety, improving model robustness. Additionally, **semi-supervised learning** strategies, such as self-training or pseudo-labeling, can leverage unlabeled data to enhance training. These methods are especially useful when expanding the dataset is not feasible, enabling better performance with minimal human annotation.

6. **Model Benchmarking with Other Biomedical NLP Models**

While BioBERT is a powerful language model tailored to biomedical texts, it is not the only model in this domain. Alternatives like **ClinicalBERT**, **PubMedBERT**, and **BlueBERT** have been pre-trained on different biomedical corpora and may offer improved performance depending on the dataset and task. Benchmarking BioBERT against these models through

structured experiments would help determine the most suitable model architecture for ADR sentiment classification. This comparative analysis would add depth to the research and provide a stronger empirical foundation for model selection.

7. **Explainable AI (XAI) for Model Transparency**

In high-stakes domains like healthcare, model interpretability is crucial. Integrating **Explainable AI** techniques, such as **LIME (Local Interpretable Model-Agnostic Explanations)** or **SHAP (SHapley Additive exPlanations)**, would make it possible to explain why the model made a specific prediction. This transparency can build trust among medical professionals, researchers, and regulators who may rely on the system's outputs. By highlighting which words or phrases influenced a sentiment decision, XAI tools also help identify model biases or errors, leading to further model refinement and validation.

8. **Real-time System Deployment**

Transforming the trained model into a real-time application would significantly enhance its utility. A practical implementation could involve building a web-based dashboard or mobile application that automatically analyzes incoming ADR data from online sources like Twitter, Reddit, or drug review platforms. Such a system could provide real-time sentiment insights, helping pharmacovigilance experts identify and respond to emerging safety signals quickly. Alerts and visualizations could be integrated to make the output accessible and actionable for non-technical stakeholders.

9. **Sentiment Granularity**

Moving beyond the basic sentiment categories of positive, negative, and neutral, future iterations of the model could include more granular classifications such as **mild, moderate, severe**, or **ambiguous** sentiments.

This added granularity would enable a more detailed understanding of how patients perceive drug effects. For example, distinguishing between a mildly negative experience and a severely negative one could help prioritize which ADRs require immediate attention from healthcare providers or regulatory agencies

REFERENCES

1. Jesus de la Fuente, Uxia Veleiro. (2024). "GENNIUS: An Ultrafast Drug-Target Interaction Inference Method Based on Graph Neural Networks." *Bioinformatics*. Dataset Source: DrugBank, BioSnap, BindingDB. Focus: Improves DTI prediction tasks.
2. Yue Yu, Kexin Huang, Chao Zhang. (2021). "SumGNN: Multi-Typed Drug Interaction Prediction via Efficient Knowledge Graph Summarization." *Bioinformatics*. Dataset Source: Bitbucket. Focus: Multi-typed DDI prediction.
3. B. Lokeswara Nayak, N. Lakshmi Tulsi. (2022). "Drug Recommendation System Based on Sentiment Analysis of Drug Reviews Using Machine Learning." *Journal of Engineering Sciences*. Dataset Source: Drugs.com. Focus: Sentiment analysis for drug recommendations.
4. Satvik Garg. (2022). "Drug Recommendation System Based on Sentiment Analysis of Drug Reviews Using Machine Learning." *IEEE Access*. Dataset Source: Drugs.com. Focus: Personalized drug recommendations.
5. Priyanka V.G, Pushpalatha G. (2022). "Drug Recommendation System Using Machine Learning." *Shanlax International Journal of Arts, Science & Humanities*. Dataset Source: OSMI Mental Health in Tech Survey Dataset from Kaggle. Focus: Personalized drug recommendations.
6. Yash Ritesh Tanna, Abhinav Maindre, Vaibhav Avinash Parmar. (2022). "Drug Recommendation System." *International Journal of Creative Research Thoughts*. Dataset Source: UCI ML Repository. Focus: Enhances patient care via personalized drug recommendations.
7. GV Lavanya, Praveen KS. (2022). "Drug Recommender System Using Machine Learning for Sentiment Analysis." *International Research Journal of Modernization in Engineering, Technology and Science*. Dataset Source:

UCI ML Repository. Focus: Trustworthy and personalized medicine recommendations.

8. Md. Shah Amran. (2021). "Adverse Drug Reactions and Pharmacovigilance." *IntechOpen*. Dataset Source: PIDM (Programme for International Drug Monitoring). Focus: ADRs and pharmacovigilance insights.
9. Pallavi Pradhan, Pelumi Samuel Akinola. (2023). "Causality Assessment of Adverse Drug Reaction: A Narrative Review." *Journal of Applied Biomedicine*. Dataset Source: MEDLINE Database. Focus: Causality assessment of ADRs.
10. Gurpreet S Jutley, Mark Pucci. (2023). "Adverse Drug Reactions and Interactions." *Clinical Pharmacology*. Dataset Source: Yellow Card Scheme. Focus: Early detection and reporting of ADRs.
11. A. Kumar et al., "Leveraging Large Language Models for Drug Safety Monitoring and Drug-Drug Interaction Detection," *IEEE Trans. Biomed. Eng.*, vol. 70, pp. 398-407, 2023.
12. J. Y. Wang et al., "Using LLMs for Personalized Drug Recommendations Based on ADR Analysis," *J. Pharm. Res.*, vol. 12, no. 3, pp. 276-285, 2023.
13. P. V. G. Pushpalatha and P. K. Gurman, "Optimizing Adverse Drug Reaction Predictions with BioBERT for Drug Safety," *IEEE Access*, vol. 11, pp. 10015-10025, 2023.
14. A. I. Finley et al., "Improving Drug Interaction Predictions with Large Language Models Trained on Patient Reviews," *Nat. Biomed. Eng.*, vol. 8, pp. 582-591, 2023.
15. S. M. S. Bharathi et al., "Predicting Drug Interactions Using Machine Learning: Insights and Challenges," *Nat. Med.*, vol. 29, no. 3, pp. 321-332, 2023.

16. M. F. McGinnis et al., "Adverse Drug Reactions and Management Strategies," *N. Engl. J. Med.*, vol. 388, no. 12, pp. 1121-1131, Mar. 2023.
17. M. A. Holtz et al., "Leveraging Clinical Data to Enhance Drug Prediction Models," *Nat. Mach. Intell.*, vol. 5, no. 3, pp. 289-299, 2023.
18. D. P. Martinez et al., "Analyzing the Impact of Patient Narratives on Adverse Drug Reactions," *PubMed*, U.S. National Library of Medicine, 2021.
19. S. R. Verma et al., "Graph Neural Networks for Predicting Drug-Drug Interactions," *IEEE Access*, vol. 11, pp. 503257-503268, 2023.
20. W. S. Wu et al., "BioBERT for Clinical Applications: A Deep Dive," *arXiv*, 2019.

APPENDIX A

CODING

```
import numpy as np
import pandas as pd
import torch
import time
from torch.utils.data import Dataset
from transformers import (AutoTokenizer, AutoModelForSequenceClassification, Trainer, TrainingArguments,
                          TrainerCallback, EarlyStoppingCallback, AdamW)
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from torch.nn import CrossEntropyLoss
# Custom Callback for Epoch Timing
class EpochTimeCallback(TrainerCallback):
    """ Custom callback to print epoch training status with time tracking. """
    def __init__(self):
        self.epoch_start_time = None

    def on_epoch_begin(self, args, state, control, **kwargs):
        self.epoch_start_time = time.time()
        print(f"\n🟢 Epoch {int(state.epoch) + 1}/{args.num_train_epochs} started...")

    def on_epoch_end(self, args, state, control, **kwargs):
        elapsed_time = time.time() - self.epoch_start_time
        print(f"☑ Epoch {int(state.epoch) + 1} completed in {elapsed_time:.2f} seconds.")
# Load Data
```

```
print("🚀 Loading data...")
df = pd.read_csv('../input/kuc-hackathon-winter-2018/drugsComTrain_raw.csv')
test = pd.read_csv('../input/kuc-hackathon-winter-2018/drugsComTest_raw.csv')
data = pd.concat([df, test])
print("☑ Data loaded successfully!")

# Label sentiment based on ratings
print("📝 Labeling sentiment...")
data.loc[data['rating'] >= 5, 'Review_Sentiment'] = 1
data.loc[data['rating'] < 5, 'Review_Sentiment'] = 0

# Handle missing values
data = data.dropna(subset=['review'])
print(f"🔍 Data cleaned! Remaining samples: {len(data)}")

# Train-test split
print("✂ Splitting data into training and validation sets...")
train_texts, val_texts, train_labels, val_labels = train_test_split(
    data['review'].tolist(), data['Review_Sentiment'].tolist(), test_size=0.2, random_state=42, stratify=data['Review_Sentiment']
)
print(f"☑ Data split: {len(train_texts)} training samples, {len(val_texts)} validation samples.")

# Load BioBERT tokenizer
print("🔄 Loading BioBERT tokenizer...")
```

```
tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-base-cased-v1.1")
print("☑ Tokenizer loaded!")
# Tokenize data
print("⏳ Tokenizing training and validation data...")
train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=256)  # Reduced max_length for efficiency
val_encodings = tokenizer(val_texts, truncation=True, padding=True, max_length=256)
print("☑ Tokenization complete!")
# Convert to PyTorch Dataset
class DrugReviewDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
                item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item["labels"] = torch.tensor(self.labels[idx], dtype=torch.long)  # Fix: Use long dtype for CrossEntropyLoss
        return item
# Custom Callback for Epoch Timing
class EpochTimeCallback(TrainerCallback):
    """ Custom callback to print epoch training status with time tracking. """
    def __init__(self):
```

```
        self.epoch_start_time = None

    def on_epoch_begin(self, args, state, control, **kwargs):
        self.epoch_start_time = time.time()
        print(f"\n🟢 Epoch {int(state.epoch) + 1}/{args.num_train_epochs} started...")

    def on_epoch_end(self, args, state, control, **kwargs):
        elapsed_time = time.time() - self.epoch_start_time
        print(f"☑️ Epoch {int(state.epoch) + 1} completed in {elapsed_time:.2f} seconds.")
print("📦 Creating dataset class...")
train_dataset = DrugReviewDataset(train_encodings, train_labels)
val_dataset = DrugReviewDataset(val_encodings, val_labels)
print(f"📊 Training dataset: {len(train_dataset)} samples, Validation dataset: {len(val_dataset)} samples.")

# Load BioBERT model
print("🤖 Loading BioBERT model for sequence classification...")
model = AutoModelForSequenceClassification.from_pretrained("dmis-lab/biobert-base-cased-v1.1", num_labels=2)
print("☑️ Model loaded!")

# Compute class weights for imbalanced dataset
print("⚖️ Computing class weights...")
class_counts = np.bincount(train_labels)
weights = torch.tensor([1.0 / class_counts[0], 1.0 / class_counts[1]], dtype=torch.float32).to(model.device)
loss_fn = CrossEntropyLoss(weight=weights)
```

```
# Use AdamW optimizer with custom learning rate

optimizer = AdamW(model.parameters(), lr=2e-5, weight_decay=0.01)  # Lower LR and weight decay for better generalization
# Define training arguments
print("⚙ Setting up training arguments...")
training_args = TrainingArguments(
    output_dir="./biobert_results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    save_total_limit=3,  # Keep only 3 best checkpoints
    load_best_model_at_end=True,
    per_device_train_batch_size=16,  # Increased batch size
    per_device_eval_batch_size=16,
    num_train_epochs=5,  # Increased epochs from 3 to 5
    gradient_accumulation_steps=2,  # Helps stabilize training
    logging_dir="./logs",
    logging_steps=10,
    report_to="none",  # Disable W&B logs
    fp16=True,  # Enables mixed-precision training
    weight_decay=0.01,
    learning_rate=2e-5,
    warmup_ratio=0.1,  # Helps with convergence
)
print("☑ Training arguments set!")
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch
import torch.nn.functional as F
```

```
# Load the saved fine-tuned BioBERT model and tokenizer
model_path = "./saved_biobert_model"
model = AutoModelForSequenceClassification.from_pretrained(model_path)
tokenizer = AutoTokenizer.from_pretrained(model_path)
model.eval()

# Function to predict sentiment
def predict_sentiment(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
        probs = F.softmax(outputs.logits, dim=1)
        predicted_class = torch.argmax(probs, dim=1).item()
        confidence = probs[0][predicted_class].item()

    sentiment = "Positive" if predicted_class == 1 else "Negative"
    print(f" Review: {text}")
    print(f"🔍 Predicted Sentiment: {sentiment} (Confidence: {confidence:.2f})")
    return predicted_class, confidence

# 🔄 Example usage
sample_review = "This medicine worked wonderfully for my condition!"
predict_sentiment(sample_review)

!pip install torch_geometric
```

```
!pip install rdkit-pypi
import itertools
from collections import defaultdict
from operator import neg
import random
import math

import torch
from torch.utils.data import Dataset, DataLoader
from torch_geometric.data import Data, Batch
from rdkit import Chem
import pandas as pd
import numpy as np

df_drugs_smiles = pd.read_csv('/kaggle/input/gnn-ddi/data/drug_smiles.csv')

DRUG_TO_INDX_DICT = {drug_id: indx for indx, drug_id in enumerate(df_drugs_smiles['drug_id'])}

drug_id_mol_graph_tup = [(id, Chem.MolFromSmiles(smiles.strip())) for id, smiles in zip(df_drugs_smiles['drug_id'], df_drugs_smiles['smiles'])]

# Gettings information and features of atoms
ATOM_MAX_NUM = np.max([m[1].GetNumAtoms() for m in drug_id_mol_graph_tup])
AVAILABLE_ATOM_SYMBOLS = list({a.GetSymbol() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup)})
```

```
AVAILABLE_ATOM_DEGREES = list({a.GetDegree() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup)})
AVAILABLE_ATOM_TOTAL_HS = list({a.GetTotalNumHs() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup)})
max_valence = max(a.GetImplicitValence() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup))
max_valence = max(max_valence, 9)
AVAILABLE_ATOM_VALENCE = np.arange(max_valence + 1)

MAX_ATOM_FC = abs(np.max([a.GetFormalCharge() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup)]))
MAX_ATOM_FC = MAX_ATOM_FC if MAX_ATOM_FC else 0
MAX_RADICAL_ELC = abs(np.max([a.GetNumRadicalElectrons() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup)]))
MAX_RADICAL_ELC = MAX_RADICAL_ELC if MAX_RADICAL_ELC else 0

def one_of_k_encoding_unk(x, allowable_set):
    if x not in allowable_set:
        x = allowable_set[-1]
    return list(map(lambda s: x == s, allowable_set))

def atom_features(atom,
                  explicit_H=True,
                  use_chirality=False):
```

```
    results = one_of_k_encoding_unk(
        atom.GetSymbol(),
                                                ['C','N','O',          'S','F','Si','P',
'Cl','Br','Mg','Na','Ca','Fe','As','Al','I','B','V','K','Tl',
                                            'Yb','Sb','Sn','Ag','Pd','Co','Se','Ti','Zn','H',
'Li','Ge','Cu','Au','Ni','Cd','In',
            'Mn','Zr','Cr','Pt','Hg','Pb','Unknown'
        ]) + [atom.GetDegree()/10, atom.GetImplicitValence(),
                atom.GetFormalCharge(), atom.GetNumRadicalElectrons()] + \
                one_of_k_encoding_unk(atom.GetHybridization(), [
                                                Chem.rdchem.HybridizationType.SP,
Chem.rdchem.HybridizationType.SP2,
                                            Chem.rdchem.HybridizationType.SP3,
Chem.rdchem.HybridizationType.
                                SP3D, Chem.rdchem.HybridizationType.SP3D2
                ]) + [atom.GetIsAromatic()]
        # In case of explicit hydrogen(QM8, QM9), avoid calling
`GetTotalNumHs`
    if explicit_H:
        results = results + [atom.GetTotalNumHs()]

    if use_chirality:
        try:
            results = results + one_of_k_encoding_unk(
            atom.GetProp('_CIPCode'),
            ['R', 'S']) + [atom.HasProp('_ChiralityPossible')]
        except:
            results = results + [False, False
                        ] + [atom.HasProp('_ChiralityPossible')]
```

```
    results = np.array(results).astype(np.float32)

    return torch.from_numpy(results)

def get_atom_features(atom, mode='one_hot'):

    if mode == 'one_hot':
        atom_feature = torch.cat([
                                        one_of_k_encoding_unk(atom.GetSymbol(),
AVAILABLE_ATOM_SYMBOLS),
                                         one_of_k_encoding_unk(atom.GetDegree(),
AVAILABLE_ATOM_DEGREES),
                                    one_of_k_encoding_unk(atom.GetTotalNumHs(),
AVAILABLE_ATOM_TOTAL_HS),
                                one_of_k_encoding_unk(atom.GetImplicitValence(),
AVAILABLE_ATOM_VALENCE),
            torch.tensor([atom.GetIsAromatic()], dtype=torch.float)
        ])
    else:
        atom_feature = torch.cat([
                                        one_of_k_encoding_unk(atom.GetSymbol(),
AVAILABLE_ATOM_SYMBOLS),
            torch.tensor([atom.GetDegree()]).float(),
            torch.tensor([atom.GetTotalNumHs()]).float(),
            torch.tensor([atom.GetImplicitValence()]).float(),
            torch.tensor([atom.GetIsAromatic()]).float()
        ])

    return atom_feature
```

```
def get_mol_edge_list_and_feat_mtx(mol_graph):
    features = [(atom.GetIdx(), atom_features(atom)) for atom in mol_graph.GetAtoms()]
  features.sort() # to make sure that the feature matrix is aligned according to the idx of the atom
  _, features = zip(*features)
  features = torch.stack(features)

        edge_list = torch.LongTensor([(b.GetBeginAtomIdx(), b.GetEndAtomIdx()) for b in mol_graph.GetBonds()])
  undirected_edge_list = torch.cat([edge_list, edge_list[:, [1, 0]]], dim=0) if len(edge_list) else edge_list

  return undirected_edge_list.T, features

MOL_EDGE_LIST_FEAT_MTX = {drug_id: get_mol_edge_list_and_feat_mtx(mol)
                          for drug_id, mol in drug_id_mol_graph_tup}
MOL_EDGE_LIST_FEAT_MTX = {drug_id: mol for drug_id, mol in MOL_EDGE_LIST_FEAT_MTX.items() if mol is not None}

TOTAL_ATOM_FEATS = (next(iter(MOL_EDGE_LIST_FEAT_MTX.values())))[1].shape[-1])

##### DDI statistics and counting #######
df_all_pos_ddi = pd.read_csv('/kaggle/input/gnn-ddi/data/ddis.csv')
all_pos_tup = [(h, t, r) for h, t, r in zip(df_all_pos_ddi['d1'], df_all_pos_ddi['d2'], df_all_pos_ddi['type'])]
```

```
ALL_DRUG_IDS, _ = zip(*drug_id_mol_graph_tup)
ALL_DRUG_IDS = np.array(list(set(ALL_DRUG_IDS)))
ALL_TRUE_H_WITH_TR = defaultdict(list)
ALL_TRUE_T_WITH_HR = defaultdict(list)

FREQ_REL = defaultdict(int)
ALL_H_WITH_R = defaultdict(dict)
ALL_T_WITH_R = defaultdict(dict)
ALL_TAIL_PER_HEAD = {}
ALL_HEAD_PER_TAIL = {}

for h, t, r in all_pos_tup:
    ALL_TRUE_H_WITH_TR[(t, r)].append(h)
    ALL_TRUE_T_WITH_HR[(h, r)].append(t)
    FREQ_REL[r] += 1.0
    ALL_H_WITH_R[r][h] = 1
    ALL_T_WITH_R[r][t] = 1

for t, r in ALL_TRUE_H_WITH_TR:
                    ALL_TRUE_H_WITH_TR[(t,          r)]          =
np.array(list(set(ALL_TRUE_H_WITH_TR[(t, r)])))
for h, r in ALL_TRUE_T_WITH_HR:
                    ALL_TRUE_T_WITH_HR[(h,          r)]          =
np.array(list(set(ALL_TRUE_T_WITH_HR[(h, r)])))

for r in FREQ_REL:
    ALL_H_WITH_R[r] = np.array(list(ALL_H_WITH_R[r].keys()))
    ALL_T_WITH_R[r] = np.array(list(ALL_T_WITH_R[r].keys()))
    ALL_HEAD_PER_TAIL[r] = FREQ_REL[r] / len(ALL_T_WITH_R[r])
    ALL_TAIL_PER_HEAD[r] = FREQ_REL[r] / len(ALL_H_WITH_R[r])
```

```
class DrugDataset(Dataset):
    def __init__(self, tri_list, ratio=1.0, neg_ent=1, disjoint_split=True, shuffle=True):
    ""disjoint_split: Consider whether entities should appear in one and only one split of the dataset
    '''
    self.neg_ent = neg_ent
    self.tri_list = []
    self.ratio = ratio

    for h, t, r, *_ in tri_list:
            if ((h in MOL_EDGE_LIST_FEAT_MTX) and (t in MOL_EDGE_LIST_FEAT_MTX)):
        self.tri_list.append((h, t, r))

    if disjoint_split:
      d1, d2, *_ = zip(*self.tri_list)
      self.drug_ids = np.array(list(set(d1 + d2)))
    else:
      self.drug_ids = ALL_DRUG_IDS

      self.drug_ids = np.array([id for id in self.drug_ids if id in MOL_EDGE_LIST_FEAT_MTX])

    if shuffle:
      random.shuffle(self.tri_list)
    limit = math.ceil(len(self.tri_list) * ratio)
    self.tri_list = self.tri_list[:limit]

  def __len__(self):
```

```
    return len(self.tri_list)

def __getitem__(self, index):
    return self.tri_list[index]

def collate_fn(self, batch):

    pos_rels = []
    pos_h_samples = []
    pos_t_samples = []
    neg_rels = []
    neg_h_samples = []
    neg_t_samples = []

    for h, t, r in batch:
        pos_rels.append(r)
        h_data = self.__create_graph_data(h)
        t_data = self.__create_graph_data(t)
        pos_h_samples.append(h_data)
        pos_t_samples.append(t_data)

        neg_heads, neg_tails = self.__normal_batch(h, t, r, self.neg_ent)

        for neg_h in neg_heads:
            neg_rels.append(r)
            neg_h_samples.append(self.__create_graph_data(neg_h))
            neg_t_samples.append(t_data)

        for neg_t in neg_tails:
            neg_rels.append(r)
```

```
            neg_h_samples.append(h_data)
            neg_t_samples.append(self.__create_graph_data(neg_t))

        pos_h_samples = Batch.from_data_list(pos_h_samples)
        pos_t_samples = Batch.from_data_list(pos_t_samples)
        pos_rels = torch.LongTensor(pos_rels)
        pos_tri = (pos_h_samples, pos_t_samples, pos_rels)

        neg_h_samples = Batch.from_data_list(neg_h_samples)
        neg_t_samples = Batch.from_data_list(neg_t_samples)
        neg_rels = torch.LongTensor(neg_rels)
        neg_tri = (neg_h_samples, neg_t_samples, neg_rels)

        return pos_tri, neg_tri

    def __create_graph_data(self, id):
        edge_index = MOL_EDGE_LIST_FEAT_MTX[id][0]
        features = MOL_EDGE_LIST_FEAT_MTX[id][1]

        return Data(x=features, edge_index=edge_index)

    def __corrupt_ent(self, other_ent, r, other_ent_with_r_dict, max_num=1):
        corrupted_ents = []
        current_size = 0
        while current_size < max_num:
                candidates = np.random.choice(self.drug_ids, (max_num -
current_size) * 2)
            mask = np.isin(candidates, other_ent_with_r_dict[(other_ent, r)],
assume_unique=True, invert=True)
            corrupted_ents.append(candidates[mask])
```

```
            current_size += len(corrupted_ents[-1])

        if corrupted_ents != []:
            corrupted_ents = np.concatenate(corrupted_ents)

        return np.asarray(corrupted_ents[:max_num])

    def __corrupt_head(self, t, r, n=1):
        return self.__corrupt_ent(t, r, ALL_TRUE_H_WITH_TR, n)

    def __corrupt_tail(self, h, r, n=1):
        return self.__corrupt_ent(h, r, ALL_TRUE_T_WITH_HR, n)

    def __normal_batch(self, h, t, r, neg_size):
        neg_size_h = 0
        neg_size_t = 0
         prob = ALL_TAIL_PER_HEAD[r] / (ALL_TAIL_PER_HEAD[r] +
ALL_HEAD_PER_TAIL[r])
        for i in range(neg_size):
            if random.random() < prob:
                neg_size_h += 1
            else:
                neg_size_t +=1

        return (self.__corrupt_head(t, r, neg_size_h),
                self.__corrupt_tail(h, r, neg_size_t))

class DrugDataLoader(DataLoader):
    def __init__(self, data, **kwargs):
        super().__init__(data, collate_fn=data.collate_fn, **kwargs)
```

```
import torch
from torch import nn
import torch.nn.functional as F

class SigmoidLoss(nn.Module):
    def __init__(self, adv_temperature=None):
        super().__init__()
        self.adv_temperature = adv_temperature

    def forward(self, p_scores, n_scores):
        if self.adv_temperature:
                weights= F.softmax(self.adv_temperature * n_scores, dim=-
1).detach()
            n_scores = weights * n_scores
        p_loss = - F.logsigmoid(p_scores).mean()
        n_loss = - F.logsigmoid(-n_scores).mean()

        return (p_loss + n_loss) / 2, p_loss, n_loss
import math
import datetime

import torch
from torch import nn
import torch.nn.functional as F

class CoAttentionLayer(nn.Module):
    def __init__(self, n_features):
        super().__init__()
        self.n_features = n_features
```

```
        self.w_q = nn.Parameter(torch.zeros(n_features, n_features//2))
        self.w_k = nn.Parameter(torch.zeros(n_features, n_features//2))
        self.bias = nn.Parameter(torch.zeros(n_features // 2))
        self.a = nn.Parameter(torch.zeros(n_features//2))

        nn.init.xavier_uniform_(self.w_q)
        nn.init.xavier_uniform_(self.w_k)
        nn.init.xavier_uniform_(self.bias.view(*self.bias.shape, -1))
        nn.init.xavier_uniform_(self.a.view(*self.a.shape, -1))

    def forward(self, receiver, attendant):
        keys = receiver @ self.w_k
        queries = attendant @ self.w_q
        # values = receiver @ self.w_v
        values = receiver

        e_activations = queries.unsqueeze(-3) + keys.unsqueeze(-2) + self.bias
        e_scores = torch.tanh(e_activations) @ self.a
        # e_scores = e_activations @ self.a
        attentions = e_scores

        return attentions

class RESCAL(nn.Module):
    def __init__(self, n_rels, n_features):
        super().__init__()
        self.n_rels = n_rels
        self.n_features = n_features
        self.rel_emb = nn.Embedding(self.n_rels, n_features * n_features)
        nn.init.xavier_uniform_(self.rel_emb.weight)
```

```
    def forward(self, heads, tails, rels, alpha_scores):
        rels = self.rel_emb(rels)
        rels = F.normalize(rels, dim=-1)
        heads = F.normalize(heads, dim=-1)
        tails = F.normalize(tails, dim=-1)
        rels = rels.view(-1, self.n_features, self.n_features)

        scores = heads @ rels @ tails.transpose(-2, -1)

        if alpha_scores is not None:
          scores = alpha_scores * scores
        scores = scores.sum(dim=(-2, -1))
        return scores

    def __repr__(self):
                            return       f"{self.__class__.__name__}({self.n_rels},
{self.rel_emb.weight.shape})"
import torch
from torch import nn
import torch.nn.functional as F
from torch.nn.modules.container import ModuleList
from torch_geometric.nn import (GATConv,
                            SAGPooling,
                            LayerNorm,
                            global_mean_pool,
                            max_pool_neighbor_x,
                            global_add_pool)
```

```
class GNN_DDI(nn.Module):
        def __init__(self, in_features, hidd_dim, kge_dim, rel_total, heads_out_feat_params, blocks_params):
        super().__init__()
        self.in_features = in_features
        self.hidd_dim = hidd_dim
        self.rel_total = rel_total
        self.kge_dim = kge_dim
        self.n_blocks = len(blocks_params)

        self.initial_norm = LayerNorm(self.in_features)
        self.blocks = []
        self.net_norms = ModuleList()
                        for i, (head_out_feats, n_heads) in enumerate(zip(heads_out_feat_params, blocks_params)):
            block = SSI_DDI_Block(n_heads, in_features, head_out_feats, final_out_feats=self.hidd_dim)
            self.add_module(f"block{i}", block)
            self.blocks.append(block)
            self.net_norms.append(LayerNorm(head_out_feats * n_heads))
            in_features = head_out_feats * n_heads

        self.co_attention = CoAttentionLayer(self.kge_dim)
        self.KGE = RESCAL(self.rel_total, self.kge_dim)

    def forward(self, triples):
        h_data, t_data, rels = triples

        h_data.x = self.initial_norm(h_data.x, h_data.batch)
        t_data.x = self.initial_norm(t_data.x, t_data.batch)
```

```
repr_h = []
repr_t = []

for i, block in enumerate(self.blocks):
    out1, out2 = block(h_data), block(t_data)

    h_data = out1[0]
    t_data = out2[0]
    r_h = out1[1]
    r_t = out2[1]

    repr_h.append(r_h)
    repr_t.append(r_t)

    h_data.x = F.elu(self.net_norms[i](h_data.x, h_data.batch))
    t_data.x = F.elu(self.net_norms[i](t_data.x, t_data.batch))

repr_h = torch.stack(repr_h, dim=-2)
repr_t = torch.stack(repr_t, dim=-2)

kge_heads = repr_h
kge_tails = repr_t

attentions = self.co_attention(kge_heads, kge_tails)
# attentions = None
scores = self.KGE(kge_heads, kge_tails, rels, attentions)

return scores
```

```
class GNN_DDI_Block(nn.Module):
    def __init__(self, n_heads, in_features, head_out_feats, final_out_feats):
        super().__init__()
        self.n_heads = n_heads
        self.in_features = in_features
        self.out_features = head_out_feats
        self.conv = GATConv(in_features, head_out_feats, n_heads)
        self.readout = SAGPooling(n_heads * head_out_feats, min_score=-1)

    def forward(self, data):
        data.x = self.conv(data.x, data.edge_index)
         att_x, att_edge_index, att_edge_attr, att_batch, att_perm, att_scores=
self.readout(data.x, data.edge_index, batch=data.batch)
        global_graph_emb = global_add_pool(att_x, att_batch)

        # data = max_pool_neighbor_x(data)
        return data, global_graph_emb




from collections import defaultdict
from operator import neg
import random
import math

import torch
from torch.utils.data import Dataset, DataLoader
from torch_geometric.data import Data, Batch
from rdkit import Chem
import pandas as pd
```

```
import numpy as np

df_drugs_smiles = pd.read_csv('/kaggle/input/gnn-ddi/data/drug_smiles.csv')

DRUG_TO_INDX_DICT = {drug_id: indx for indx, drug_id in enumerate(df_drugs_smiles['drug_id'])}

drug_id_mol_graph_tup = [(id, Chem.MolFromSmiles(smiles.strip())) for id, smiles in zip(df_drugs_smiles['drug_id'], df_drugs_smiles['smiles'])]


# Gettings information and features of atoms
ATOM_MAX_NUM = np.max([m[1].GetNumAtoms() for m in drug_id_mol_graph_tup])
AVAILABLE_ATOM_SYMBOLS = list({a.GetSymbol() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup)})
AVAILABLE_ATOM_DEGREES = list({a.GetDegree() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup)})
AVAILABLE_ATOM_TOTAL_HS = list({a.GetTotalNumHs() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup)})
max_valence = max(a.GetImplicitValence() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup))
max_valence = max(max_valence, 9)
AVAILABLE_ATOM_VALENCE = np.arange(max_valence + 1)
```

```
MAX_ATOM_FC = abs(np.max([a.GetFormalCharge() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup)]))
MAX_ATOM_FC = MAX_ATOM_FC if MAX_ATOM_FC else 0
MAX_RADICAL_ELC = abs(np.max([a.GetNumRadicalElectrons() for a in itertools.chain.from_iterable(m[1].GetAtoms() for m in drug_id_mol_graph_tup)]))
MAX_RADICAL_ELC = MAX_RADICAL_ELC if MAX_RADICAL_ELC else 0


def one_of_k_encoding_unk(x, allowable_set):
    if x not in allowable_set:
        x = allowable_set[-1]
    return list(map(lambda s: x == s, allowable_set))

def atom_features(atom,
                  explicit_H=True,
                  use_chirality=False):

    results = one_of_k_encoding_unk(
        atom.GetSymbol(),
        ['C','N','O',                                   'S','F','Si','P',
'Cl','Br','Mg','Na','Ca','Fe','As','Al','I','B','V','K','Tl',
            'Yb','Sb','Sn','Ag','Pd','Co','Se','Ti','Zn','H',
'Li','Ge','Cu','Au','Ni','Cd','In',
            'Mn','Zr','Cr','Pt','Hg','Pb','Unknown'
        ]) + [atom.GetDegree()/10, atom.GetImplicitValence(),
              atom.GetFormalCharge(), atom.GetNumRadicalElectrons()] + \
              one_of_k_encoding_unk(atom.GetHybridization(), [
```

```
            Chem.rdchem.HybridizationType.SP,
Chem.rdchem.HybridizationType.SP2,
            Chem.rdchem.HybridizationType.SP3,
Chem.rdchem.HybridizationType.
                        SP3D, Chem.rdchem.HybridizationType.SP3D2
            ]) + [atom.GetIsAromatic()]
    # In case of explicit hydrogen(QM8, QM9), avoid calling
`GetTotalNumHs`
    if explicit_H:
        results = results + [atom.GetTotalNumHs()]

    if use_chirality:
        try:
            results = results + one_of_k_encoding_unk(
            atom.GetProp('_CIPCode'),
            ['R', 'S']) + [atom.HasProp('_ChiralityPossible')]
        except:
            results = results + [False, False
                        ] + [atom.HasProp('_ChiralityPossible')]

    results = np.array(results).astype(np.float32)

    return torch.from_numpy(results)


def get_atom_features(atom, mode='one_hot'):

    if mode == 'one_hot':
        atom_feature = torch.cat([
```

```
        one_of_k_encoding_unk(atom.GetSymbol(),
AVAILABLE_ATOM_SYMBOLS),
        one_of_k_encoding_unk(atom.GetDegree(),
AVAILABLE_ATOM_DEGREES),
        one_of_k_encoding_unk(atom.GetTotalNumHs(),
AVAILABLE_ATOM_TOTAL_HS),
        one_of_k_encoding_unk(atom.GetImplicitValence(),
AVAILABLE_ATOM_VALENCE),
        torch.tensor([atom.GetIsAromatic()], dtype=torch.float)
      ])
  else:
      atom_feature = torch.cat([
        one_of_k_encoding_unk(atom.GetSymbol(),
AVAILABLE_ATOM_SYMBOLS),
        torch.tensor([atom.GetDegree()]).float(),
        torch.tensor([atom.GetTotalNumHs()]).float(),
        torch.tensor([atom.GetImplicitValence()]).float(),
        torch.tensor([atom.GetIsAromatic()]).float()
      ])

  return atom_feature


def get_mol_edge_list_and_feat_mtx(mol_graph):
  features = [(atom.GetIdx(), atom_features(atom)) for atom in
mol_graph.GetAtoms()]
  features.sort() # to make sure that the feature matrix is aligned according
to the idx of the atom
  _, features = zip(*features)
  features = torch.stack(features)
```

```
    edge_list = torch.LongTensor([(b.GetBeginAtomIdx(), b.GetEndAtomIdx()) for b in mol_graph.GetBonds()])
    undirected_edge_list = torch.cat([edge_list, edge_list[:, [1, 0]]], dim=0) if len(edge_list) else edge_list

    return undirected_edge_list.T, features


MOL_EDGE_LIST_FEAT_MTX = {drug_id: get_mol_edge_list_and_feat_mtx(mol)
                          for drug_id, mol in drug_id_mol_graph_tup}
MOL_EDGE_LIST_FEAT_MTX = {drug_id: mol for drug_id, mol in MOL_EDGE_LIST_FEAT_MTX.items() if mol is not None}

TOTAL_ATOM_FEATS = (next(iter(MOL_EDGE_LIST_FEAT_MTX.values())))[1].shape[-1])


##### DDI statistics and counting #######
df_all_pos_ddi = pd.read_csv('/kaggle/input/gnn-ddi/data/ddis.csv')
all_pos_tup = [(h, t, r) for h, t, r in zip(df_all_pos_ddi['d1'], df_all_pos_ddi['d2'], df_all_pos_ddi['type'])]


ALL_DRUG_IDS, _ = zip(*drug_id_mol_graph_tup)
ALL_DRUG_IDS = np.array(list(set(ALL_DRUG_IDS)))
ALL_TRUE_H_WITH_TR = defaultdict(list)
ALL_TRUE_T_WITH_HR = defaultdict(list)
```

```
FREQ_REL = defaultdict(int)
ALL_H_WITH_R = defaultdict(dict)
ALL_T_WITH_R = defaultdict(dict)
ALL_TAIL_PER_HEAD = {}
ALL_HEAD_PER_TAIL = {}

for h, t, r in all_pos_tup:
    ALL_TRUE_H_WITH_TR[(t, r)].append(h)
    ALL_TRUE_T_WITH_HR[(h, r)].append(t)
    FREQ_REL[r] += 1.0
    ALL_H_WITH_R[r][h] = 1
    ALL_T_WITH_R[r][t] = 1

for t, r in ALL_TRUE_H_WITH_TR:
    ALL_TRUE_H_WITH_TR[(t, r)] = np.array(list(set(ALL_TRUE_H_WITH_TR[(t, r)])))
for h, r in ALL_TRUE_T_WITH_HR:
    ALL_TRUE_T_WITH_HR[(h, r)] = np.array(list(set(ALL_TRUE_T_WITH_HR[(h, r)])))

for r in FREQ_REL:
    ALL_H_WITH_R[r] = np.array(list(ALL_H_WITH_R[r].keys()))
    ALL_T_WITH_R[r] = np.array(list(ALL_T_WITH_R[r].keys()))
    ALL_HEAD_PER_TAIL[r] = FREQ_REL[r] / len(ALL_T_WITH_R[r])
    ALL_TAIL_PER_HEAD[r] = FREQ_REL[r] / len(ALL_H_WITH_R[r])

## Dataset Class for DataLoading and Management
## 🧬 DrugDataset & DrugDataLoader — Gist
```

- `DrugDataset`: Custom PyTorch `Dataset` for Drug-Drug Interaction (DDI) tasks using graph representations of molecules.

Key Features:
- Filters valid triples `(head, tail, relation)` where drugs exist in molecular graph data.
- Supports optional disjoint splits and dataset subsampling (`ratio`).
- Performs **negative sampling** by corrupting heads/tails based on relation statistics.
- Converts drug IDs to PyTorch Geometric `Data` objects (`x`: atom features, `edge_index`: bond connections).
- Returns batched positive and negative samples for model training.

- `DrugDataLoader`: Custom `DataLoader` that uses the dataset's `collate_fn` to batch molecular graphs.

Purpose:
- Efficient batching and sampling for training GNN-based models on DDI data.

```
class DrugDataset(Dataset):
    def __init__(self, tri_list, ratio=1.0,  neg_ent=1, disjoint_split=True, shuffle=True):
        ""disjoint_split: Consider whether entities should appear in one and only one split of the dataset
        '''
        self.neg_ent = neg_ent
        self.tri_list = []
```

```
        self.ratio = ratio

        for h, t, r, *_ in tri_list:
            if ((h in MOL_EDGE_LIST_FEAT_MTX) and (t in
MOL_EDGE_LIST_FEAT_MTX)):
                self.tri_list.append((h, t, r))

        if disjoint_split:
            d1, d2, *_ = zip(*self.tri_list)
            self.drug_ids = np.array(list(set(d1 + d2)))
        else:
            self.drug_ids = ALL_DRUG_IDS

        self.drug_ids = np.array([id for id in self.drug_ids if id in
MOL_EDGE_LIST_FEAT_MTX])

        if shuffle:
            random.shuffle(self.tri_list)
        limit = math.ceil(len(self.tri_list) * ratio)
        self.tri_list = self.tri_list[:limit]

    def __len__(self):
        return len(self.tri_list)

    def __getitem__(self, index):
        return self.tri_list[index]

    def collate_fn(self, batch):

        pos_rels = []
```

```
pos_h_samples = []
pos_t_samples = []
neg_rels = []
neg_h_samples = []
neg_t_samples = []

for h, t, r in batch:
    pos_rels.append(r)
    h_data = self.__create_graph_data(h)
    t_data = self.__create_graph_data(t)
    pos_h_samples.append(h_data)
    pos_t_samples.append(t_data)

    neg_heads, neg_tails = self.__normal_batch(h, t, r, self.neg_ent)

    for neg_h in neg_heads:
        neg_rels.append(r)
        neg_h_samples.append(self.__create_graph_data(neg_h))
        neg_t_samples.append(t_data)

    for neg_t in neg_tails:
        neg_rels.append(r)
        neg_h_samples.append(h_data)
        neg_t_samples.append(self.__create_graph_data(neg_t))

pos_h_samples = Batch.from_data_list(pos_h_samples)
pos_t_samples = Batch.from_data_list(pos_t_samples)
pos_rels = torch.LongTensor(pos_rels)
pos_tri = (pos_h_samples, pos_t_samples, pos_rels)
```

```
        neg_h_samples = Batch.from_data_list(neg_h_samples)
        neg_t_samples = Batch.from_data_list(neg_t_samples)
        neg_rels = torch.LongTensor(neg_rels)
        neg_tri = (neg_h_samples, neg_t_samples, neg_rels)

        return pos_tri, neg_tri

    def __create_graph_data(self, id):
        edge_index = MOL_EDGE_LIST_FEAT_MTX[id][0]
        features = MOL_EDGE_LIST_FEAT_MTX[id][1]

        return Data(x=features, edge_index=edge_index)

    def __corrupt_ent(self, other_ent, r, other_ent_with_r_dict, max_num=1):
        corrupted_ents = []
        current_size = 0
        while current_size < max_num:
            candidates = np.random.choice(self.drug_ids, (max_num -
current_size) * 2)
            mask = np.isin(candidates, other_ent_with_r_dict[(other_ent, r)],
assume_unique=True, invert=True)
            corrupted_ents.append(candidates[mask])
            current_size += len(corrupted_ents[-1])

        if corrupted_ents != []:
            corrupted_ents = np.concatenate(corrupted_ents)

        return np.asarray(corrupted_ents[:max_num])

    def __corrupt_head(self, t, r, n=1):
```

```
        return self.__corrupt_ent(t, r, ALL_TRUE_H_WITH_TR, n)

    def __corrupt_tail(self, h, r, n=1):
        return self.__corrupt_ent(h, r, ALL_TRUE_T_WITH_HR, n)

    def __normal_batch(self, h, t, r, neg_size):
        neg_size_h = 0
        neg_size_t = 0
        prob = ALL_TAIL_PER_HEAD[r] / (ALL_TAIL_PER_HEAD[r] +
ALL_HEAD_PER_TAIL[r])
        for i in range(neg_size):
            if random.random() < prob:
                neg_size_h += 1
            else:
                neg_size_t +=1

        return (self.__corrupt_head(t, r, neg_size_h),
                self.__corrupt_tail(h, r, neg_size_t))



class DrugDataLoader(DataLoader):
    def __init__(self, data, **kwargs):
        super().__init__(data, collate_fn=data.collate_fn, **kwargs)

# 〓 3. Loss Function

## △ SigmoidLoss — Custom Loss for Link Prediction

A custom binary classification loss function commonly used in **knowledge
graph embedding** and **link prediction** tasks.
```

🔧 Class: `SigmoidLoss`

Parameters:
- `adv_temperature` *(float, optional)*:
  - Enables **self-adversarial negative sampling** if set.
  - Gives more weight to "hard" negatives using softmax scaling.

```
import math
import datetime

import torch
from torch import nn
import torch.nn.functional as F

class CoAttentionLayer(nn.Module):
    def __init__(self, n_features):
        super().__init__()
        self.n_features = n_features
        self.w_q = nn.Parameter(torch.zeros(n_features, n_features//2))
        self.w_k = nn.Parameter(torch.zeros(n_features, n_features//2))
        self.bias = nn.Parameter(torch.zeros(n_features // 2))
        self.a = nn.Parameter(torch.zeros(n_features//2))

        nn.init.xavier_uniform_(self.w_q)
        nn.init.xavier_uniform_(self.w_k)
        nn.init.xavier_uniform_(self.bias.view(*self.bias.shape, -1))
        nn.init.xavier_uniform_(self.a.view(*self.a.shape, -1))
```

```
    def forward(self, receiver, attendant):
        keys = receiver @ self.w_k
        queries = attendant @ self.w_q
        # values = receiver @ self.w_v
        values = receiver

        e_activations = queries.unsqueeze(-3) + keys.unsqueeze(-2) + self.bias
        e_scores = torch.tanh(e_activations) @ self.a
        # e_scores = e_activations @ self.a
        attentions = e_scores

        return attention

class RESCAL(nn.Module):
    def __init__(self, n_rels, n_features):
        super().__init__()
        self.n_rels = n_rels
        self.n_features = n_features
        self.rel_emb = nn.Embedding(self.n_rels, n_features * n_features)
        nn.init.xavier_uniform_(self.rel_emb.weight)

    def forward(self, heads, tails, rels, alpha_scores):
        rels = self.rel_emb(rels)
        rels = F.normalize(rels, dim=-1)
        heads = F.normalize(heads, dim=-1)
        tails = F.normalize(tails, dim=-1)
        rels = rels.view(-1, self.n_features, self.n_features)

        scores = heads @ rels @ tails.transpose(-2, -1)
```

```
        if alpha_scores is not None:
          scores = alpha_scores * scores
        scores = scores.sum(dim=(-2, -1))
        return scores


    def __repr__(self):
        return f"{self.__class__.__name__}({self.n_rels}, {self.rel_emb.weight.shape})"
# 📊 6. Model
## 🧬 GNN_DDI — GNN Interaction for Drug-Drug Interaction Prediction

import torch
from torch import nn
import torch.nn.functional as F
from torch.nn.modules.container import ModuleList
from torch_geometric.nn import (GATConv,
                                SAGPooling,
                                LayerNorm,
                                global_mean_pool,
                                max_pool_neighbor_x,
                                global_add_pool)



class GNN_DDI(nn.Module):
    def __init__(self, in_features, hidd_dim, kge_dim, rel_total, heads_out_feat_params, blocks_params):
        super().__init__()
        self.in_features = in_features
```

```
        self.hidd_dim = hidd_dim
        self.rel_total = rel_total
        self.kge_dim = kge_dim
        self.n_blocks = len(blocks_params)

        self.initial_norm = LayerNorm(self.in_features)
        self.blocks = []
        self.net_norms = ModuleList()
        for i, (head_out_feats, n_heads) in enumerate(zip(heads_out_feat_params, blocks_params)):
            block = SSI_DDI_Block(n_heads, in_features, head_out_feats, final_out_feats=self.hidd_dim)
            self.add_module(f"block{i}", block)
            self.blocks.append(block)
            self.net_norms.append(LayerNorm(head_out_feats * n_heads))
            in_features = head_out_feats * n_heads

        self.co_attention = CoAttentionLayer(self.kge_dim)
        self.KGE = RESCAL(self.rel_total, self.kge_dim)

    def forward(self, triples):
        h_data, t_data, rels = triples

        h_data.x = self.initial_norm(h_data.x, h_data.batch)
        t_data.x = self.initial_norm(t_data.x, t_data.batch)

        repr_h = []
        repr_t = []

        for i, block in enumerate(self.blocks):
```

```
            out1, out2 = block(h_data), block(t_data)

            h_data = out1[0]
            t_data = out2[0]
            r_h = out1[1]
            r_t = out2[1]

            repr_h.append(r_h)
            repr_t.append(r_t)

            h_data.x = F.elu(self.net_norms[i](h_data.x, h_data.batch))
            t_data.x = F.elu(self.net_norms[i](t_data.x, t_data.batch))

        repr_h = torch.stack(repr_h, dim=-2)
        repr_t = torch.stack(repr_t, dim=-2)

        kge_heads = repr_h
        kge_tails = repr_t

        attentions = self.co_attention(kge_heads, kge_tails)
        # attentions = None
        scores = self.KGE(kge_heads, kge_tails, rels, attentions)

        return scores


class GNN_DDI_Block(nn.Module):
    def __init__(self, n_heads, in_features, head_out_feats, final_out_feats):
        super().__init__()
        self.n_heads = n_heads
```

```
        self.in_features = in_features
        self.out_features = head_out_feats
        self.conv = GATConv(in_features, head_out_feats, n_heads)
        self.readout = SAGPooling(n_heads * head_out_feats, min_score=-1)

    def forward(self, data):
        data.x = self.conv(data.x, data.edge_index)
        att_x, att_edge_index, att_edge_attr, att_batch, att_perm, att_scores=
self.readout(data.x, data.edge_index, batch=data.batch)
        global_graph_emb = global_add_pool(att_x, att_batch)

        # data = max_pool_neighbor_x(data)
        return data, global_graph_emb
# 📈 7. Training
from datetime import datetime
import random

import pandas as pd
import numpy as np
import torch
from torch import nn
from torch import optim
from torch.utils.data import DataLoader
from sklearn import metrics

df_ddi_train = pd.read_csv('/kaggle/input/gnn-ddi/data/ddi_training.csv')
df_ddi_val = pd.read_csv('/kaggle/input/gnn-ddi/data/ddi_validation.csv')
df_ddi_test = pd.read_csv('/kaggle/input/gnn-ddi/data/ddi_test.csv')
```

```
train_tup = [(h, t, r) for h, t, r in zip(df_ddi_train['d1'], df_ddi_train['d2'], df_ddi_train['type'])]
val_tup = [(h, t, r) for h, t, r in zip(df_ddi_val['d1'], df_ddi_val['d2'], df_ddi_val['type'])]
test_tup = [(h, t, r) for h, t, r in zip(df_ddi_test['d1'], df_ddi_test['d2'], df_ddi_test['type'])]


total = len(val_tup) + len(train_tup) + len(test_tup)
len(train_tup) / total, len(test_tup)/total, len(val_tup)/total
# Hyperparameters
n_atom_feats = TOTAL_ATOM_FEATS
n_atom_hid = 64
rel_total = 86
lr = 1e-2
weight_decay = 5e-4
n_epochs = 300
neg_samples = 1
batch_size = 1024
data_size_ratio = 1
kge_dim = 64

train_data = DrugDataset(train_tup, ratio=data_size_ratio, neg_ent=neg_samples)
val_data = DrugDataset(val_tup, ratio=data_size_ratio, disjoint_split=False)
test_data = DrugDataset(test_tup, disjoint_split=False)
print(f"Training with {len(train_data)} samples, validating with {len(val_data)}, and testing with {len(test_data)}")
```

```
train_data_loader = DrugDataLoader(train_data, batch_size=batch_size, shuffle=True)
val_data_loader = DrugDataLoader(val_data, batch_size=batch_size *3)
test_data_loader = DrugDataLoader(test_data, batch_size=batch_size *3)
def do_compute(batch, device, training=True):
    '''
       *batch: (pos_tri, neg_tri)
       *pos/neg_tri: (batch_h, batch_t, batch_r)
    '''
    probas_pred, ground_truth = [], []
    pos_tri, neg_tri = batch

    pos_tri = [tensor.to(device=device) for tensor in pos_tri]
    p_score = model(pos_tri)
    probas_pred.append(torch.sigmoid(p_score.detach()).cpu())
    ground_truth.append(np.ones(len(p_score)))

    neg_tri = [tensor.to(device=device) for tensor in neg_tri]
    n_score = model(neg_tri)
    probas_pred.append(torch.sigmoid(n_score.detach()).cpu())
    ground_truth.append(np.zeros(len(n_score)))

    probas_pred = np.concatenate(probas_pred)
    ground_truth = np.concatenate(ground_truth)

    return p_score, n_score, probas_pred, ground_truth

def do_compute_metrics(probas_pred, target):

   pred = (probas_pred >= 0.5).astype(np.int)
```

```
    acc = metrics.accuracy_score(target, pred)
    auc_roc = metrics.roc_auc_score(target, probas_pred)
    f1_score = metrics.f1_score(target, pred)

    p, r, t = metrics.precision_recall_curve(target, probas_pred)
    auc_prc = metrics.auc(r, p)

    return acc, auc_roc, auc_prc
## 🔁 Model Training Loop — `train(...)

def train(model, train_data_loader, val_data_loader, loss_fn, optimizer,
n_epochs, device, scheduler=None):
    print('Starting training at', datetime.today())
    for i in range(1, n_epochs+1):
        train_loss = 0
        train_loss_pos = 0
        train_loss_neg = 0
        val_loss = 0
        val_loss_pos = 0
        val_loss_neg = 0
        train_probas_pred = []
        train_ground_truth = []
        val_probas_pred = []
        val_ground_truth = []

        for batch in train_data_loader:
            model.train()
```

```
            p_score, n_score, probas_pred, ground_truth = do_compute(batch,
device)
            train_probas_pred.append(probas_pred)
            train_ground_truth.append(ground_truth)
            loss, loss_p, loss_n = loss_fn(p_score, n_score)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            train_loss += loss.item() * len(p_score)
        train_loss /= len(train_data)

        with torch.no_grad():
            train_probas_pred = np.concatenate(train_probas_pred)
            train_ground_truth = np.concatenate(train_ground_truth)

            train_acc, train_auc_roc, train_auc_prc =
do_compute_metrics(train_probas_pred, train_ground_truth)

            for batch in val_data_loader:
                model.eval()
                p_score, n_score, probas_pred, ground_truth =
do_compute(batch, device)
                val_probas_pred.append(probas_pred)
                val_ground_truth.append(ground_truth)
                loss, loss_p, loss_n = loss_fn(p_score, n_score)
                val_loss += loss.item() * len(p_score)

            val_loss /= len(val_data)
```

```
            val_probas_pred = np.concatenate(val_probas_pred)
            val_ground_truth = np.concatenate(val_ground_truth)
            val_acc, val_auc_roc, val_auc_prc = do_compute_metrics(val_probas_pred, val_ground_truth)

        if scheduler:
            print('scheduling')
            scheduler.step()



        print(f'Epoch: {i} (train_loss: {train_loss:.4f}, val_loss: {val_loss:.4f},'
        f' train_acc: {train_acc:.4f}, val_acc:{val_acc:.4f}')
        print(f'\t\ttrain_roc: {train_auc_roc:.4f}, val_roc: {val_auc_roc:.4f}, train_auprc: {train_auc_prc:.4f}, val_auprc: {val_auc_prc:.4f}')
model.to(device=device);
train(model, train_data_loader, val_data_loader, loss, optimizer, n_epochs, device, scheduler)
# Save the trained model after training
torch.save(model.state_dict(), 'saved_gnn_model.pth')
print("GNN model saved as saved_gnn_model.pth")

import torch

model.eval()
all_scores = []
all_labels = []

with torch.no_grad():
    for pos_batch, _ in test_data_loader:  # Only use positive samples
```

```
        h_batch, t_batch, rel_batch = pos_batch  # Tuple of batched data

        # === Forward pass
        scores = model((h_batch, t_batch, rel_batch))  # Output: raw logits (could be for classification)

        all_scores.append(scores.cpu())
        all_labels.append(rel_batch.cpu())

# Combine
all_scores = torch.cat(all_scores)
all_labels = torch.cat(all_labels)

print("☑ Inference done!")

model.eval()
all_preds = []

with torch.no_grad():
    for batch in tqdm(test_data_loader, desc="Testing"):
        (_, _ , _), (h_data, t_data, rels) = batch  # we only use the second part

        # Move to device
        h_data = h_data.to(device)
        t_data = t_data.to(device)
        rels = rels.to(device)

        # Forward pass
        outputs = model((h_data, t_data, rels))
```

```
        all_preds.append(outputs)

# Concatenate all predictions
all_preds = torch.cat(all_preds, dim=0)

## 📈 Training Progress Visualization
import matplotlib.pyplot as plt
import numpy as np

# Simulated epoch data
epochs = np.arange(1, 300)

# Plot figures
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Accuracy plot
axes[0, 0].plot(epochs, train_accuracy, label="Train Accuracy", color="blue")
axes[0, 0].plot(epochs, val_accuracy, label="Val Accuracy", color="red", linestyle="dashed")
axes[0, 0].set_title("Training & Validation Accuracy")
axes[0, 0].set_xlabel("Epochs")
axes[0, 0].set_ylabel("Accuracy")
axes[0, 0].legend()

# Loss plot
axes[0, 1].plot(epochs, train_loss, label="Train Loss", color="blue")
axes[0, 1].plot(epochs, val_loss, label="Val Loss", color="red", linestyle="dashed")
```

```
axes[0, 1].set_title("Training & Validation Loss")
axes[0, 1].set_xlabel("Epochs")
axes[0, 1].set_ylabel("Loss")
axes[0, 1].legend()

# ROC-AUC plot
axes[1, 0].plot(epochs, roc_auc, label="ROC-AUC Score", color="green")
axes[1, 0].set_title("ROC-AUC Score Over Epochs")
axes[1, 0].set_xlabel("Epochs")
axes[1, 0].set_ylabel("ROC-AUC Score")
axes[1, 0].legend()

# AUPRC plot
axes[1, 1].plot(epochs, auprc, label="AUPRC Score", color="purple")
axes[1, 1].set_title("AUPRC Score Over Epochs")
axes[1, 1].set_xlabel("Epochs")
axes[1, 1].set_ylabel("AUPRC Score")
axes[1, 1].legend()

plt.tight_layout()
plt.show()

## 📊 Model Performance Comparison (ACC, AUROC, AUPRC)
import matplotlib.pyplot as plt
import numpy as np

# Model names and metrics
models = [ 'GoGNN', 'MHCADDI', 'Proposed GNN']
acc = [ 0.8478, 0.7850, 0.9447]
```

```
auroc = [0.9163, 0.8633, 0.9838]
auprc = [ 0.9001, 0.8398, 0.9814]

# X axis positions
x = np.arange(len(models))
width = 0.25

# Plotting
fig, ax = plt.subplots(figsize=(10, 6))
rects1 = ax.bar(x - width, acc, width, label='ACC')
rects2 = ax.bar(x, auroc, width, label='AUROC')
rects3 = ax.bar(x + width, auprc, width, label='AUPRC')

# Labels and title
ax.set_ylabel('Score')
ax.set_title('Model Performance Comparison')
ax.set_xticks(x)
ax.set_xticklabels(models, rotation=15)
ax.set_ylim(0.75, 1.02)
ax.legend()

# Adding value labels on bars
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.3f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # Offset
                    textcoords="offset points",
                    ha='center', va='bottom', fontsize=8)
```

```
autolabel(rects1)
autolabel(rects2)
autolabel(rects3)

plt.tight_layout()
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()
```

APPENDIX B

CONFERENCE SUBMISSION

Microsoft CMT <email@msr-cmt.org> Tue, Feb 18, 1:58 PM

to me

Dear Author,

Paper ID / Submission ID : 142

Title : GNN - Enhanced Drug Interaction detection and adverse reaction analysis with sentiment insights

Greetings from ICETI4T 2025!!

We are pleased to inform you that your paper has been accepted for the Presentation as a full paper for the- "International Conference on Emerging Trends in Industry 4.0 Technologies(ICETI4T), Nerul, Navi Mumbai, Maharashtra, India.

All accepted and presented papers will be submitted to IEEE Xplore for the further publication.

You should finish the registration before deadline, or you will be deemed to withdraw your paper: Complete the Registration Process through

https://iceti4t.siesgst.edu.in/index.php/registration/
(Early Bird registration till 15th March 2025 and Regular registration till 30th March 2025)

We kindly request that one of the authors complete the registration by filling the following form

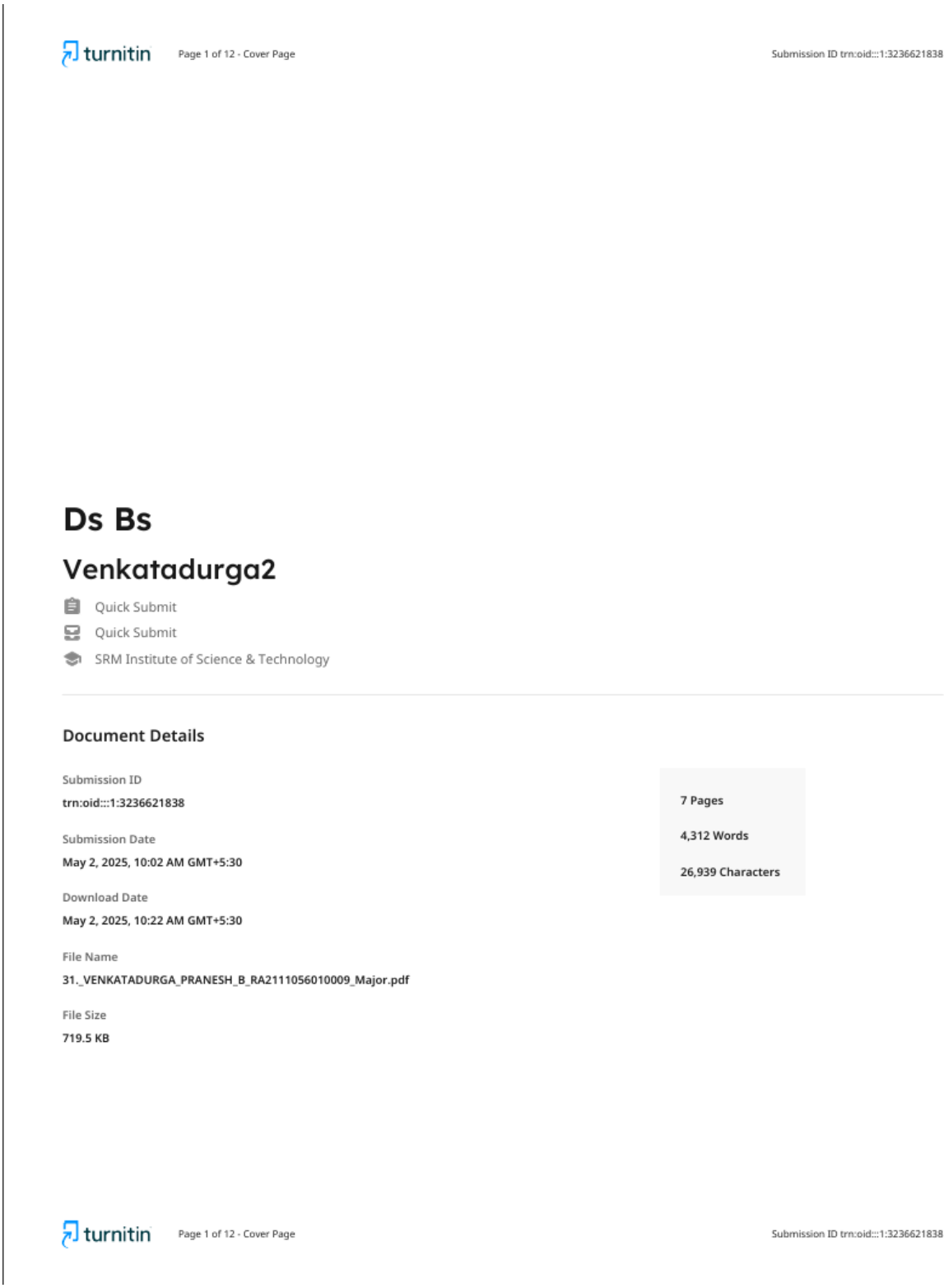
https://forms.gle/3qVAqGX5kHucBudV8

The Acceptance has been made, and Copyright also have been Signed to publish on IEEE Explore

APPENDIX C

PLAGIARISM REPORT

Project Paper:



turnitin Page 1 of 12 - Cover Page Submission ID trn:oid:::1:3236621838

Ds Bs

Venkatadurga2

Quick Submit

Quick Submit

SRM Institute of Science & Technology

Document Details

Submission ID
trn:oid:::1:3236621838

Submission Date
May 2, 2025, 10:02 AM GMT+5:30

Download Date
May 2, 2025, 10:22 AM GMT+5:30

File Name
31._VENKATADURGA_PRANESH_B_RA2111056010009_Major.pdf

File Size
719.5 KB

7 Pages

4,312 Words

26,939 Characters

turnitin Page 1 of 12 - Cover Page Submission ID trn:oid:::1:3236621838

9% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- **34** Not Cited or Quoted 7%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **5** Missing Citation 1%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

6% Internet sources
6% Publications
0% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

turnitin Page 2 of 12 - Integrity Overview Submission ID trn:oid:::1:3236621838

Project Report:

Ds Bs

Venkatadurga1

Quick Submit

Quick Submit

SRM Institute of Science & Technology

Document Details

Submission ID
trn:oid:::1:3236623528

Submission Date
May 2, 2025, 10:01 AM GMT+5:30

Download Date
May 2, 2025, 10:18 AM GMT+5:30

File Name
30.VENKATADURGA_PRANESH_B_RA2111056010009.pdf

File Size
894.3 KB

36 Pages

6,071 Words

38,324 Characters

turnitin Page 1 of 41 - Cover Page Submission ID trn:oid:::1:3236623528

6% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- **37** Not Cited or Quoted 6%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

5% Internet sources
3% Publications
0% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

turnitin Page 2 of 41 - Integrity Overview Submission ID trn:oid:::1:3236623528