# A Framework for Automated Low Latency Adverse Drug Reaction Reporting Using Crowdsourced Reporting and Graph Theory Analysis

Dr Jyotirmay Kirtania, Professor of Anesthesiology, Critical Care & Pain
MPMMCC & HBCH (Tata Memorial Centre) Varanasi, India
Email: jyotirmay@mpmmcc.tmc.gov.in
https://orcid.org/0000-0002-4426-6877
https://mpmmcc.tmc.gov.in/
Manuscript version 1.5, 15Sep2024

## Abstract:

Adverse Drug Reactions (ADRs) are a leading cause of hospital admissions and healthcare costs. Traditional methods of ADR reporting often rely on post-marketing surveillance, and manual reporting of ADRs to the local or national pharmacovigilance agencies for causality assessment and final reporting to the WHO. High-income countries have their own national (i.e., USFDA) and regional (i.e., European Medicines Agency / EMA) pharmacovigilance agencies. However, this process is slow and inefficient. This article proposes a novel framework for integrating ADR detection into clinical workflows using Electronic Medical Record (EMR) systems, crowdsourced reporting from patients and healthcare professionals, and graph theory for generating automated ADR signals and reports to the local or national pharmacovigilance agencies. The system leverages automated data collection from EMRs (drug prescriptions, clinical notes) by EMR data scraping, integrating ADR dictionaries and drug databases to automate the generation of ranked ADR signals. By applying graph theory, the system filters and upranks connections between drugs and ADRs, considering the temporal relationship between drug administration and ADR occurrence. This automated approach offers a significant improvement in ADR reporting, enabling faster detection and more accurate predictions. Methodologies, framework visualizations and python code snippets are included to aid implementation.

# Introduction

The International Conference on Harmonization of Technical Requirements for Registration of Pharmaceuticals for Human Use, of which the World Health Organization (WHO) and the United States Food and Drug Administration (FDA) are members, defines an ADR as "A response to a drug which is noxious and unintended, and which occurs at doses normally used for prophylaxis, diagnosis, or therapy of disease or the modification of physiologic function." [1]

Adverse drug reactions (ADRs) present a significant challenge in the healthcare sector. In 2022, worldwide, there were over 1.25 million serious adverse events reported and nearly 175,000 deaths. [2] There are 6 emergency department (ED) visits for therapeutic and nontherapeutic medication harms per 1,000 patients, and about 38% of such visits subsequently require hospitalization.[3] Additionally, in 3 out of every 1000 hospital admissions, a patient dies due to an ADR.[4]

ADR-related healthcare costs are significant—and preventable. In the United States and Europe, the financial burden is estimated at $30.1 billion US dollars and €79 billion euros, respectively [5, 6]. Despite improved access to medicines, the data on the impact of ADRs in low-and middle-income countries is scarce and very likely underestimated.

Traditional ADR reporting systems, such as those implemented by the USFDA and other regulatory agencies like the World Health Organization (WHO) and the European Medicines Agency (EMA), often suffer from underreporting and delayed identification of drug-related adverse events. A study assessing ADRs in hospitalized patients revealed that physicians often overlook a significant portion of these reactions.[7] Too large a share of medicines risk management remains limited to signal detection in big ADR databases (USFDA, EMA, WHO, etc.) This resource allocation is antiquated and applied statistical signal detection methodologies have reached their limits of usefulness. [8]

There is an unmet need to develop an automated, low latency ADR signal reporting system.

Recent studies [9, 10, 11, 12] have demonstrated the potential of computational models to detect and predict ADR signals from EMR data using advanced machine learning algorithms and graph neural networks.

# Methods:

## Framework Overview

Input Data Sources:

The system integrates data from multiple sources

1. EMR Prescription and Drug Administration Data can be extracted from hospital EMR systems at predefined intervals.

2. Barcode Scanning: Real-time drug administration tracking via barcodes of drugs administered by nurses in the hospital (portable barcode scanner connected to the EMR), or patients or caregivers at home (mobile phone's barcode scanner application).
3. ADR Dictionary: A comprehensive list of generic and proprietary drug names with associated adverse drug reactions based on the drug's Summary of Product Characteristics (SPC) published by the manufacturers. This ADR Dictionary shall be continuously updated.
4. Crowdsourced Clinical Notes: Adverse symptoms and signs collected from clinical notes recorded by healthcare professionals in healthcare settings. Adverse symptoms reported by patients and home caregivers through mobile application or phone call to the physician or nurse.

Data Collection & Processing:
The system periodically scrapes prescription and drug administration data from EMRs. For real-time administration, barcodes are scanned during drug administration, linking the drug name and timestamp to the patient's EMR record. Additionally, clinical notes are analyzed using natural language processing (NLP) to extract potential ADRs based on a predefined ADR keyword dictionary.

Graph Theory-Based ADR Detection:
The framework constructs a bipartite graph with two sets of nodes:

a) Drug Nodes: Each node represents a prescribed drug.
b) ADR Nodes: Each node represents a reported ADR.
c) Edges between these nodes are weighted based on several factors:
d) Timing: If the ADR occurred after the drug administration, the edge weight is higher. Pre-existing conditions or symptoms (reported before drug administration) are down ranked.
e) Frequency: The more frequent the ADR reports for a particular drug, the higher the edge weight.

The following algorithms are employed:

a) Degree Centrality: Measures the strength of connections (edges) between drugs and ADRs.
b) PageRank: A modified version ranks drug-ADR pairs based on the weighted edges, prioritizing ADRs with the strongest links.
c) Downranking Pre-existing Symptoms: To reduce false ADR signals, the system checks whether a symptom or clinical sign was recorded prior to drug administration. If present, these connections are assigned lower edge weights. This prevents

pre-existing conditions from being incorrectly attributed as ADRs, improving the system's accuracy.

Sample Code Implementation: The system is built using Python's Flask for backend operations and a frontend dashboard.

Backend: The Flask API handles data collection and analysis. Key functions include:

i) EMR Data Scraping: Extracts prescription and drug administration data from the EMR.
ii) Barcode Scanning Integration: A REST API endpoint collects real-time barcode data.
iii) Graph Theory Analysis: Using NetworkX, the API constructs a graph and applies ranking algorithms.

Workflow Explanation:

1. Recording ADR Report: When an ADR is reported (either by a clinician or a patient), the system now also records the time when the symptom/sign first appeared (i.e., symptom_timestamp).
2. Check Against Prescription Time: Before connecting the ADR to the drug, the system checks if the symptom was recorded before or after the drug administration.
3. If the symptom was already present before the drug was prescribed/administered, it's less likely to be caused by the drug, and the connection is assigned a low weight (e.g., 0.1).
4. If the symptom appeared after drug administration, the connection is given a normal weight (e.g., 1.0).
5. Ranking ADRs: The connections between drugs and ADRs are then ranked based on the edge weights in the graph. Pre-existing symptoms will naturally result in lower-ranked connections, while new symptoms post-administration will rank higher.
6. Graph Theory-Based Ranking: The graph theory-based algorithm (e.g., degree centrality) still runs, but it now takes into account the timing of symptoms. Drug-ADR pairs where symptoms appeared after drug administration will rank higher, while those with pre-existing symptoms will rank lower.

Various Graph Theory Models can be considered for this ADR signaling framework:

1. Bipartite Graph Model
    a. A bipartite graph consists of two distinct sets of nodes. In the context of ADR detection, one set of nodes represents drugs, and the other represents adverse drug reactions (ADRs). Edges are drawn between drugs and ADRs based on reports or observations of adverse reactions after drug administration.

   b. Each edge connects a drug to a reported ADR. This model is well-suited for visualizing the relationships between prescribed drugs and the adverse reactions they may cause.
   c. This model is ideal for tracking the connections between drugs and ADRs while maintaining the separation between drug nodes and ADR nodes, making it easy to map and analyze the relationships.
2. Weighted Graph
   a. A weighted graph assigns a numerical value (weight) to the edges between nodes. In this ADR detection system, the weights represent the strength of the connection between a drug and an ADR. This weight could be influenced by the frequency of reports, severity of the ADR, or the time elapsed between drug administration and the occurrence of the ADR.
   b. Weights are assigned to reflect the frequency or severity of ADRs associated with each drug. For instance, higher weights are assigned to ADRs that appear after drug administration, and lower weights to ADRs reported before.
   c. This model helps quantify the strength of the association between a drug and an ADR, improving the accuracy of ranking ADR signals based on their relevance.
3. Directed Graph (Digraph)
   a. In directed graphs, edges between nodes have a direction, indicating a causal relationship. For ADR detection, a directed edge would point from a drug node to an ADR node if the ADR was reported after the drug administration.
   b. Directed edges capture the temporal relationship between drug administration and ADR occurrence, allowing for the modeling of cause-and-effect relationships.
   c. This is particularly valuable when analyzing time-sensitive data, ensuring that only ADRs that occur post-administration are ranked as significant.
4. Degree Centrality
   a. Degree centrality measures the number of edges connected to a node. For ADR detection, it could measure how many ADRs are linked to a specific drug or how many drugs are linked to a specific ADR.
   b. Drugs with a high degree of centrality are those with frequent ADR associations, and ADRs with high centrality affect many different drugs.
   c. Effective for ranking drugs based on their overall connection to ADRs, helping identify high-risk medications or highly common ADRs.
5. PageRank Algorithm
   a. PageRank is a graph algorithm used to rank the importance of nodes in a graph based on their connections. In the ADR detection system, it can rank drugs and ADRs by not only counting the number of connections but also considering the weight and significance of these connections.

   b. Drugs and ADRs are ranked based on their overall impact in the graph, where connections with high weights and frequent reports are given more importance.
   c. This is particularly useful in prioritizing which drugs are most likely to cause significant ADRs and which ADRs require immediate attention.
6. Graph Attention Networks (GAT)
   a. Graph Attention Networks (GAT) use attention mechanisms to focus on the most relevant parts of the graph. In the context of ADR detection, the network would focus on the most important connections between drugs and ADRs based on factors such as frequency, timing, and severity.
   b. GAT assigns different levels of attention to different edges, allowing the model to prioritize the most important drug-ADR relationships for more accurate predictions.
   c. GAT can dynamically identify and prioritize critical ADRs, improving the system's ability to focus on the most significant drug-ADR interactions.
7. Graph Isomorphism Networks (GIN)
   a. Graph Isomorphism Networks (GIN) are used to extract patterns and features from graph data, often employed in deep learning for classification tasks. In ADR detection, GIN could help classify different types of ADRs based on their association with various drugs.
   b. GIN could be applied to predict new ADRs by learning from past associations, providing a powerful tool for proactive drug safety monitoring.
   c. This approach is useful for predicting future ADRs based on historical data and recognizing patterns that suggest high-risk drugs or combinations.
8. Community Detection Algorithms
   a. Community detection algorithms, such as Louvain or Girvan-Newman, identify clusters or communities within a graph where nodes are more densely connected to each other than to the rest of the graph. This would group drugs that have similar ADR profiles or ADRs that frequently co-occur with the same set of drugs.
   b. Group drugs that cause similar ADRs or identify clusters of ADRs that are common across several drugs.
   c. Useful for identifying patterns and correlations in drug safety data that may not be apparent through individual reports, helping to cluster drugs or ADRs for further investigation.
9. Shortest Path Algorithms
   a. Algorithms like Dijkstra's or Bellman-Ford calculate the shortest path between nodes in a graph. For ADR detection, these algorithms can be used to determine the shortest time interval between drug administration and the appearance of an ADR.

    b. Calculate the shortest time between drug administration and ADR reporting, which could help prioritize urgent or acute ADRs for further investigation.
    c. Allows the system to focus on ADRs that occur soon after drug administration, identifying fast-acting adverse reactions that may require immediate clinical attention.
10. Multi-Relational Graphs
    a. Multi-relational graphs allow edges to represent different types of relationships between nodes. In the ADR detection system, one type of edge could represent direct drug-ADR interactions, while others could represent indirect influences, such as comorbidities or polypharmacy.
    b. Multi-relational graphs could model more complex interactions in healthcare, including how other factors like patient history or interactions between multiple drugs contribute to ADRs.
    c. This model would enable more detailed analyses of complex drug-ADR relationships, taking into account a variety of factors that influence ADR occurrence.

In this framework, to optimize the balance between efficacy, latency, signal to noise ratio of the ADR signaling system the following graph models and algorithms are implemented:

1. A bipartite weighted graph is implemented as the base structure for the ADR detection system, as it allows for clear separation between drugs and ADRs while incorporating the strength of connections.
2. Degree centrality and PageRank algorithms are employed for ranking the most significant ADRs and drugs, focusing on the frequency and severity of ADR reports.
3. Incorporating Graph Attention Networks (GAT) and Graph Isomorphism Networks (GIN) enhanced the system's ability to predict new ADRs based on historical patterns, helping to proactively detect potential risks before they become widespread.
4. Directed graphs and shortest path algorithms ensure that ADRs occurring shortly after drug administration are prioritized, reducing the likelihood of misclassification of pre-existing conditions as ADRs.
5. Implementing community detection algorithms help identify clusters of drugs and ADRs, offering insights into common drug interactions and patterns that may be indicative of broader safety concerns.
6. Training and Validation of Machine Learning Models (GAT, GIN) will be required before real world deployment.

By integrating these graph theory models and algorithms, this framework of ADR detection system will provide a robust framework for low latency monitoring, analysis, and prediction of adverse drug reactions in a clinical setting.

Integrating the Proposed APIs with Hospital EMR Systems (e.g., HL7 FHIR): For effective deployment of the ADR detection system in real-world hospital environments, it is essential to integrate the API endpoints (for EMR data scraping and barcode scanning) with the hospital's existing Electronic Medical Record (EMR) systems. A widely used standard for healthcare data interoperability is HL7 FHIR (Fast Healthcare Interoperability Resources). FHIR uses standard RESTful API practices with support for OAuth 2.0 for authentication and authorization. It is essential to ensure that patient data is securely accessed using encrypted HTTPS connections; and Role-based access control (RBAC) is implemented to ensure only authorized personnel can access sensitive medication and ADR data. For secure communication between the ADR system and the EMR, the hospital's FHIR server should support SMART on FHIR authentication, which provides an additional layer of security.

The framework implementation guide is clarified by the following figures and python code snippets: Figure 1, Figure 2, Figure 3, Figure 4, Python code snippet 1, Python code snippet 2, Python code snippet 3, Python code snippet 4, Python code snippet 5, Python code snippet 6

## Use Case Scenarios

This framework is designed to be deployed across multiple healthcare settings:

1. Domiciliary Care: Patients report ADRs via a mobile application, and these reports are linked to their prescribed drugs in the EMR. The system can automatically filter and rank ADRs based on their temporal relationship to the drug prescription.
2. Outpatient Department (OPD): During follow-up visits, physicians or patients report ADRs, and the system automatically links these reports to the patient's medication history. The system can flag high-risk ADRs based on the timing and frequency of reports.
3. Daycare: In daycare settings, where drugs are administered for short-term treatments, the system can track real-time administration via barcode scanning and reports ADRs in real time. Delayed ADRs can be reported via telemedicine platforms.
4. Inpatient Department (IPD): In inpatient settings, real-time drug administration is recorded, and the system can continuously monitor for ADR signals. This allows for near-real-time detection and ranking of ADRs, providing clinicians with immediate insights.
5. Acute Care (Intensive Care Units, Emergency Room, Operating Rooms): In acute care settings, such as the ICU or operating rooms, the system can track critical drugs and their immediate effects. Rapid detection of ADRs in these settings is crucial for patient safety, and the system ensures that fast-acting ADRs are prioritized for review.

## Use Case Flow for API Integration:

A. Prescription Data Flow:

1. The ADR system periodically queries the hospital's FHIR-enabled EMR system to retrieve patient prescriptions using the `MedicationRequest` resource.
2. The ADR system parses the prescription data and stores it locally for further analysis.
3. Whenever an ADR is reported, the ADR system correlates the ADR with the patient's prescriptions and medication history.

B. Drug Administration Data Flow (real-time or near-real-time):

1. When a drug is administered to a patient (either manually or via barcode scanning), the event is recorded in the FHIR-based EMR system using the `MedicationAdministration` resource.
2. The ADR detection system queries this resource periodically or in real-time to gather drug administration events.
3. This data is integrated into the system to monitor for potential ADRs occurring after the drug administration.

### Data Privacy:

By ensuring that the ADR detection system complies with global data privacy regulations such as HIPAA and GDPR, and by implementing best practices for encryption, access control, and anonymization, patient data can be safeguarded. Additionally, patient trust can be maintained by securing informed consent and offering transparency in the data collection and analysis process.

## Conclusion

This novel system offers an automated scalable solution for integrating ADR detection into healthcare workflows. By leveraging near-real-time data collection, crowdsourced reporting, and advanced graph theory algorithms, this system can significantly improve the coverage and time latency of ADR reporting. The system's ability to predict ADRs before they become widespread, combined with its capability to identify complex interactions between drugs and ADRs, can make it a valuable tool for improving drug safety. Future work could focus on enhancing the system's natural language processing capabilities for more sophisticated ADR detection and expanding the system to detect interactions between multiple drugs.

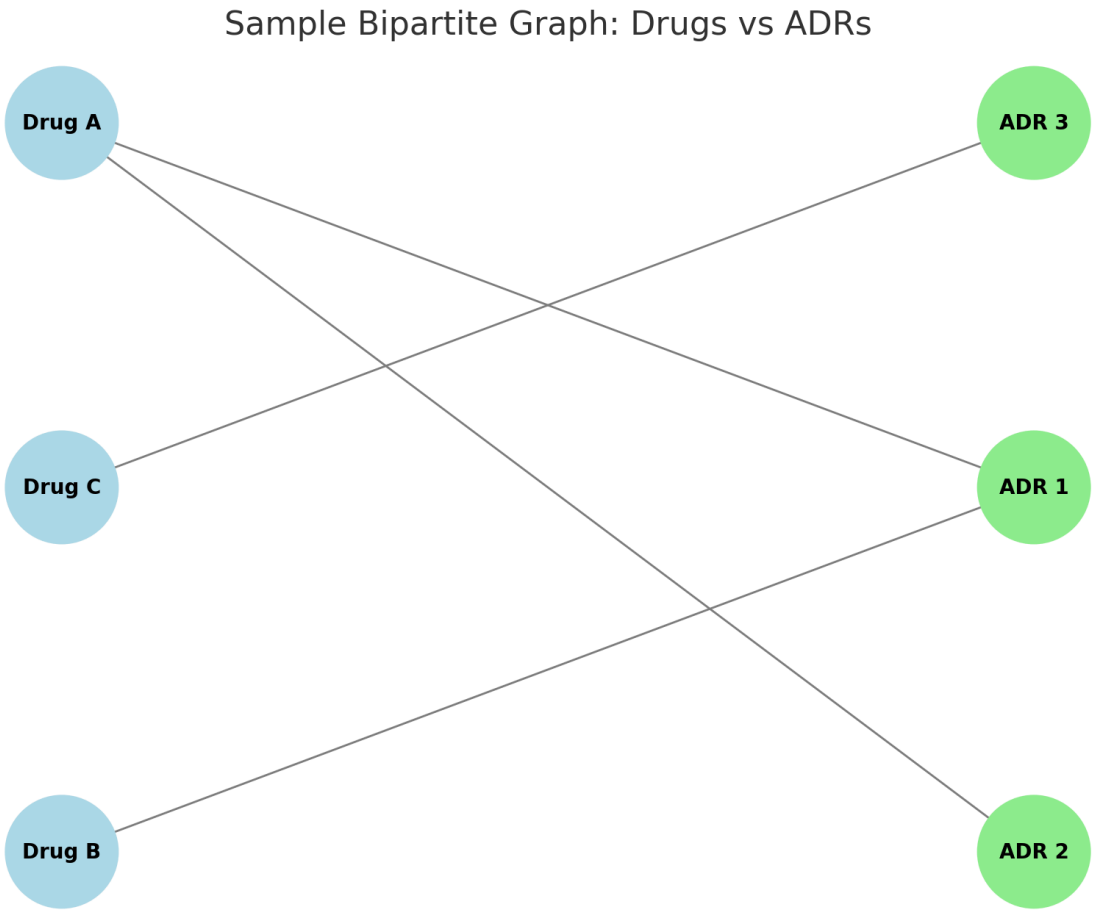
## References:

1. International drug monitoring: the role of national centres. Report of a WHO meeting. World Health Organ Tech Rep Ser. 1972;498:1-25

2. Lazarou J, Pomeranz BH, Corey PN. Incidence of adverse drug reactions in hospitalized patients: a meta-analysis of prospective studies. JAMA. 1998 Apr 15;279(15):1200-5.
3. Budnitz DS, Shehab N, Lovegrove MC, Geller AI, Lind JN, Pollock DA. US Emergency Department Visits Attributed to Medication Harms, 2017-2019. JAMA. 2021 Oct 05;326(13):1299-1309.
4. Manasse HR. Medication use in an imperfect world: drug misadventuring as an issue of public policy, Part 1. Am J Hosp Pharm. 1989 May;46(5):929-44.
5. Sultana J, Cutroneo P, Trifiro G. Clinical and economic burden of adverse drug reactions. J Pharmacol Pharmacother. 2013;4(Suppl1):S73–S77.
6. Commission of the European Communities. Commission staff working document Annex 2 of the Report on the impact assessment of strengthening and rationalizing EU Pharmacovigilance. 2008.
7. Classen DC, Pestotnik SL, Evans RS, Burke JP. Computerized surveillance of adverse drug events in hospital patients. JAMA. 1991 Nov 27;266(20):2847-51.
8. Le Louët H, Pitts PJ. Twenty-First Century Global ADR Management: A Need for Clarification, Redesign, and Coordinated Action. Ther Innov Regul Sci. 2023 Jan;57(1):100-103.
9. Ying Zheng and Shibo Xu. 2024. Predicting Frequencies of Drug Side Effects Using Graph Attention Networks with Multiple Features. In Bioinformatics Research and Applications: 20th International Symposium, ISBRA 2024, Kunming, China, July 19–21, 2024, Proceedings, Part II. Springer-Verlag, Berlin, Heidelberg, 14–25. https://doi.org/10.1007/978-981-97-5131-0_2
10. Li S, Zhang L, Wang L, Ji J, He J, Zheng X, Cao L, Li K. BiMPADR: A Deep Learning Framework for Predicting Adverse Drug Reactions in New Drugs. Molecules. 2024; 29(8):1784. https://doi.org/10.3390/molecules29081784
11. Yang J, Hu Z, Zhang L, Peng B. Predicting Drugs Suspected of Causing Adverse Drug Reactions Using Graph Features and Attention Mechanisms. Pharmaceuticals. 2024; 17(7):822. https://doi.org/10.3390/ph17070822
12. Kwak H, Lee M, Yoon S, Chang J, Park S, Jung K. Drug-Disease Graph: Predicting Adverse Drug Reaction Signals via Graph Neural Network with Clinical Data. Advances in Knowledge Discovery and Data Mining. 2020 Apr 17;12085:633–44. https://doi.org/10.1007/978-3-030-47436-2_48

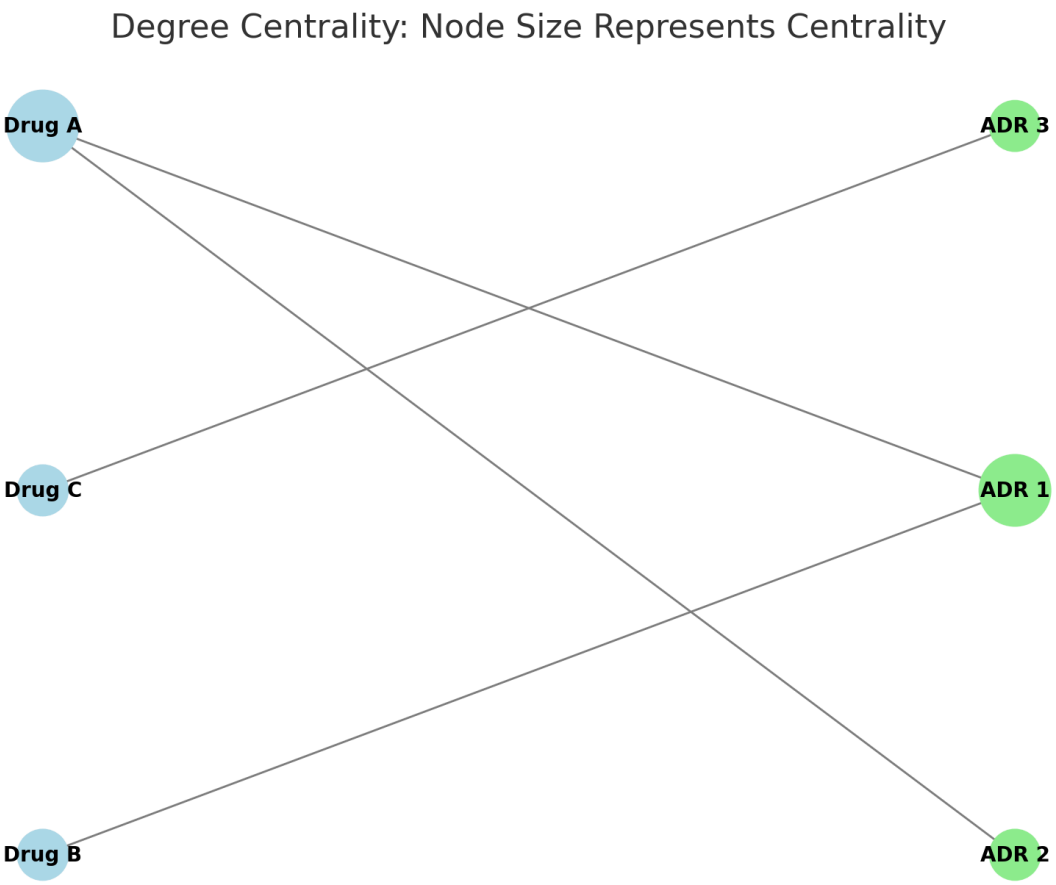
# Figures and Code Snippets

**Figure 1: Visualization of a Bipartite graph, showing the relationship between drugs and their associated ADRs (Adverse Drug Reactions).**



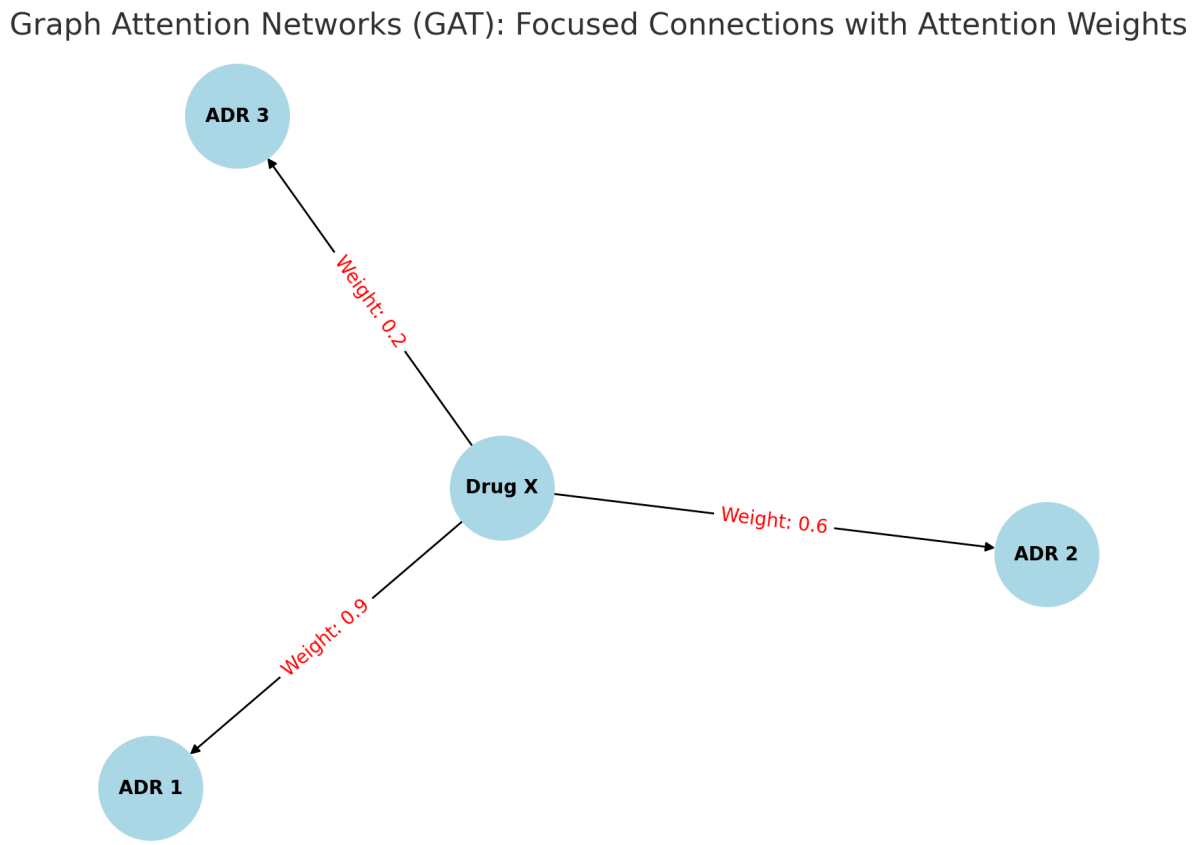

*Note: The blue nodes represent drugs, while the green nodes represent ADRs. The edges indicate the connections between specific drugs and the ADRs they cause.*

**Figure 2: Visualization of Degree Centrality in a bipartite graph of drugs and ADRs**



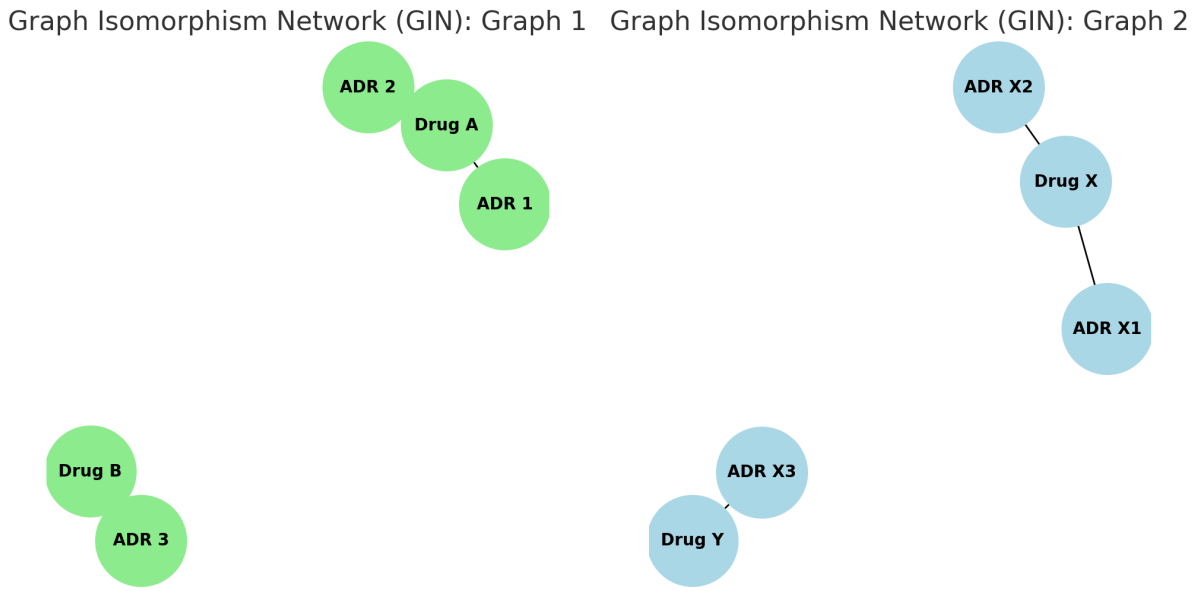

*Note: The node size in the bipartite graph represents its centrality. Nodes with more connections (higher degree centrality) are larger.*

**Figure 3: Visualization of Graph Attention Network (GAT) of drugs and ADRs**




*Note: This diagram demonstrates how attention weights are assigned to edges, focusing on the most important connections between a drug and ADRs.*

**Figure 4: Visualization of Graph Isomorphism Networks (GIN) of drugs and ADRs**




*Note: This diagram illustrates two isomorphic graphs (with similar structures) to show how GIN recognizes patterns in graph structures.*

**Python sample code 1: Backend API using Flask to handle data collection and analysis with end points for EMR Data Scraping and Barcode Scanning Integration**

```
# Sample code for the backend API using Flask to handle data collection and analysis.
# The API will include endpoints for EMR Data Scraping and Barcode Scanning Integration.

from flask import Flask, jsonify, request
import datetime

app = Flask(__name__)

# Sample data structure for storing scraped EMR data and barcode scanning records
prescriptions = []
barcode_scans = []

# Endpoint for EMR Data Scraping (simulating data extraction from EMR)
@app.route('/emr_data_scraping', methods=['POST'])
def emr_data_scraping():
        # Simulate receiving data from the EMR system (POST request)
        data = request.json
        prescription_data = {
        "patient_id": data['patient_id'],
        "drug_name": data['drug_name'],
        "prescription_time": datetime.datetime.now(),
        "doctor_id": data['doctor_id']
```

```
        }
        prescriptions.append(prescription_data)

        return jsonify({"status": "Prescription data received", "data": prescription_data}),
201

# Endpoint for Barcode Scanning Integration
@app.route('/barcode_scan', methods=['POST'])
def barcode_scan():
        # Simulate receiving barcode scan data
        data = request.json
        scan_data = {
        "drug_name": data['drug_name'],
        "scan_time": datetime.datetime.now(),
        "administered_by": data['administered_by'],  # nurse, caregiver, etc.
        "patient_id": data['patient_id']
        }
        barcode_scans.append(scan_data)

        return jsonify({"status": "Barcode scan data recorded", "data": scan_data}), 201

# Endpoint to retrieve all prescription data
@app.route('/prescriptions', methods=['GET'])
def get_prescriptions():
        return jsonify({"prescriptions": prescriptions})

# Endpoint to retrieve all barcode scans
@app.route('/barcode_scans', methods=['GET'])
def get_barcode_scans():
        return jsonify({"barcode_scans": barcode_scans})

# Start the Flask application
if __name__ == '__main__':
        app.run(debug=True)
```

Notes:

`/emr_data_scraping`**:** This endpoint simulates data scraping from an EMR system. It accepts POST requests with JSON payload containing patient information, drug name, and prescription details.

```
{
        "patient_id": "12345",
        "drug_name": "Drug A",
        "doctor_id": "D001"
}
```

`/barcode_scan`: This endpoint simulates receiving real-time barcode scan data (e.g., when a nurse or caregiver scans a drug before administering it). It accepts POST requests and stores the scan information, including drug name and the person administering the drug.

```
{
        "drug_name": "Drug A",
        "administered_by": "Nurse B",
        "patient_id": "12345"
}
```

`/prescriptions`: This GET endpoint returns all stored prescription data.

```
{
        "prescriptions": [
        {
        "patient_id": "12345",
        "drug_name": "Drug A",
        "prescription_time": "2024-09-14T10:30:00",
        "doctor_id": "D001"
        },
        {
        "patient_id": "67890",
        "drug_name": "Drug B",
        "prescription_time": "2024-09-14T12:00:00",
        "doctor_id": "D002"
        }
        ]
}
```

**Python code snippet 2: Ranking ADRs Using Degree Centrality and PageRank**

```
import networkx as nx

# Create a graph (can be bipartite or standard graph)
G = nx.Graph()

# Sample Data: Adding drugs (nodes) and ADRs (nodes) with edges (drug-ADR
relationships)
G.add_edges_from([
        ("Drug A", "ADR 1"),
        ("Drug A", "ADR 2"),
        ("Drug B", "ADR 1"),
        ("Drug C", "ADR 3"),
])
```

```
# 1. Calculate Degree Centrality: Rank nodes based on the number of direct connections
degree_centrality = nx.degree_centrality(G)
print("Degree Centrality Ranking:", degree_centrality)

# 2. Calculate PageRank: Rank nodes by their overall influence, considering edge weights
# Weights in this case can represent factors like frequency of ADR reports or severity
page_rank = nx.pagerank(G, alpha=0.85)
print("PageRank Ranking:", page_rank)

# 3. Downranking Pre-existing Conditions:
# Example logic - downrank edges (Drug-ADR relationships) if the ADR was reported
before the drug was administered
for drug, adr in G.edges():
        if adr_was_reported_before(drug, adr):  # Custom logic to check condition
        G[drug][adr]['weight'] = 0.1  # Assign lower weight
        else:
        G[drug][adr]['weight'] = 1.0  # Assign normal weight

# 4. Recalculate PageRank based on new weights after downranking
weighted_page_rank = nx.pagerank(G, alpha=0.85, weight='weight')
print("Weighted PageRank Ranking:", weighted_page_rank)

# Helper function (Pseudo-code):
def adr_was_reported_before(drug, adr):
        # Logic to determine if ADR was present before drug administration
        # For example, compare timestamps of drug administration and ADR report
        # Return True if ADR was pre-existing, otherwise False
        pass
```

**Python code snippet 3: Shortest Path Algorithm (Dijkstra's Algorithm)**

```
import networkx as nx

# Create a weighted graph
G = nx.Graph()

# Add nodes and weighted edges (drug-ADR relationships with timing weights)
G.add_weighted_edges_from([
        ("Drug A", "ADR 1", 2.0),  # Drug A causes ADR 1 after 2 hours
        ("Drug A", "ADR 2", 1.0),  # Drug A causes ADR 2 after 1 hour
        ("Drug B", "ADR 3", 3.0),  # Drug B causes ADR 3 after 3 hours
])

# Find the shortest path between Drug A and ADR 1 based on weight (time)
shortest_path = nx.dijkstra_path(G, "Drug A", "ADR 1")
print("Shortest Path (based on time):", shortest_path)
```

**Python code snippet 4: Community Detection Algorithm (Louvain)**

```
import networkx as nx
import community as community_louvain

# Create a graph
G = nx.Graph()

# Add edges (drug-ADR relationships)
G.add_edges_from([
        ("Drug A", "ADR 1"),
        ("Drug A", "ADR 2"),
        ("Drug B", "ADR 1"),
        ("Drug C", "ADR 3"),
])

# Apply Louvain community detection
partition = community_louvain.best_partition(G)

# Output the detected communities
print("Communities:", partition)
```

**Python code snippet 5: API Integration of FHIR-Based Prescription Data Retrieval**

```
import requests

FHIR_BASE_URL = "https://hospital-emr-system.com/fhir"
PATIENT_ID = "12345"

# FHIR endpoint to retrieve prescription (MedicationRequest) data for a patient
response = requests.get(f"{FHIR_BASE_URL}/MedicationRequest?patient={PATIENT_ID}")
prescription_data = response.json()

# Process the retrieved FHIR data and extract drug prescriptions
for entry in prescription_data['entry']:
        medication_request = entry['resource']
        drug_name = medication_request['medicationCodeableConcept']['text']
        doctor_id = medication_request['requester']['identifier']['value']

        # Store the prescription details in the ADR detection system
        prescription = {
        "patient_id": PATIENT_ID,
        "drug_name": drug_name,
        "prescription_time": medication_request['authoredOn'],
        "doctor_id": doctor_id
        }
```

```
        prescriptions.append(prescription)

print("Prescription Data:", prescriptions)
```

*Note: This integration will allow the ADR detection system to automatically pull prescription data from the hospital's FHIR server and store it for ADR analysis.*

**Python code snippet 6: API Integration for FHIR-Based Barcode Scanning Data Retrieval**

```
import requests

# FHIR endpoint to retrieve medication administration data for a patient
response =
requests.get(f"{FHIR_BASE_URL}/MedicationAdministration?patient={PATIENT_ID}")
administration_data = response.json()

# Process the retrieved FHIR data and extract administration details
for entry in administration_data['entry']:
        medication_admin = entry['resource']
        drug_name = medication_admin['medicationCodeableConcept']['text']
        admin_time = medication_admin['effectivePeriod']['start']
        administered_by = medication_admin['performer'][0]['actor']['reference']

        # Store the administration details in the ADR detection system
        barcode_scan = {
        "patient_id": PATIENT_ID,
        "drug_name": drug_name,
        "scan_time": admin_time,
        "administered_by": administered_by
        }
        barcode_scans.append(barcode_scan)

print("Barcode Scan Data:", barcode_scans)
```