Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів класифікації шахрайських блокчейн транзакцій
(тема)

Виконав:
студент 2 курсу, групи СШІм-22-2
Трубчанін О. В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)
Керівник доц. Турута О.П.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Трубчаніну Олексію Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів класифікації шахрайських блокчейн транзакцій

затверджена наказом університету від 1 квітня 20 24 р. № 260Ст
2. Термін подання студентом роботи до екзаменаційної комісії 6 червня 20 24 р.
3. Вихідні дані до роботи науково-технічні публікації, дані Інтернет-джерел та відомих наукових проєктів щодо тематики дослідження

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі
2) Виявлення вимог до розробленої технології
3) Опис прийнятих проектних рішень

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	01.04.2024	виконано
2	Аналіз завдання та аналогів з теми роботи	09.04.2024	виконано
3	Розробка технічних вимог до програми	10.04.2024 – 19.04.2024	виконано
4	Структурне проектування	20.04.2024 – 28.04.2024	виконано
5	Розробка алгоритмів машинного навчання	29.04.2024 – 03.05.2024	виконано
6	Вибір середовища розробки та інструментів	04.05.2024 – 09.05.2024	виконано
7	Реалізація системи	10.05.2024 – 19.05.2024	виконано
8	Тестування системи	20.05.2024	виконано
9	Розробка «Посібника користувача»	21.05.2024	виконано
10	Оформлення пояснювальної записки	21.05.2024 – 29.05.2024	виконано
11	Представлення на рецензування	02.06.2024	виконано
12	Представлення кваліфікаційної роботи до ЕК	06.06.2024	виконано

Дата видачі завдання 1 квітня 20 24 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Турута О.П.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 94 с., 13 рис., 1 дод., 14 джерел.

АНАЛІЗ ШАХРАЙСТВА У БЛОКЧЕЙН-ТРАНЗАКЦІЯХ, БАЗА ДАНИХ, ІНТЕРФЕЙС КОРИСТУВАЧА, МЕТОДИ МАШИННОГО НАВЧАННЯ, PYTHON, POSTGRESQL, TENSORFLOW.

Об’єкт розробки – процес виявлення шахрайських транзакцій у блокчейн-мережі.

Предмет розробки – алгоритми класифікації транзакцій у блокчейн-мережі як шахрайських або не шахрайських.

Мета роботи – розробка системи для автоматичного виявлення шахрайських транзакцій у блокчейні.

Методи дослідження – аналіз літератури, аналіз існуючих технологій для виявлення шахрайства, дослідження методів машинного навчання, моделювання системи, концептуальне та фізичне проектування бази даних.

Результати роботи – система для автоматичного виявлення шахрайських транзакцій у блокчейні.

Об’єкт дослідження та предмет дослідження – об'єктом дослідження є процес виявлення шахрайських транзакцій у блокчейн-мережі, а предметом дослідження є алгоритми машинного навчання та методи класифікації, які дозволяють автоматично ідентифікувати шахрайські транзакції. Охоплення різних аспектів, таких як обробка великих обсягів даних, інтеграція з блокчейн-мережами та забезпечення безпеки даних, є ключовим для розробки цієї системи.

ABSTRACT

Master’s thesis contains: 94 pp., 13 fig., 1 ann., 14 references.

DATABASE, FRAUD ANALYSIS IN BLOCKCHAIN TRANSACTIONS, MACHINE LEARNING METHODS, PYTHON, POSTGRESQL, TENSORFLOW, USER INTERFACE.

Object of development – the process of detecting fraudulent transactions in a blockchain network.

The subject of development is algorithms for classifying transactions in a blockchain network as fraudulent or not fraudulent.

Purpose – to develop a system for automatic detection of fraudulent transactions in the blockchain.

Research methods – literature analysis, analysis of existing technologies for fraud detection, research of machine learning methods, system modeling, conceptual and physical design of the database.

Results – a system for automatic detection of fraudulent transactions in the blockchain.

The object of research and subject of research – the object of research is the process of detecting fraudulent transactions in the blockchain network, and the subject of research is machine learning algorithms and classification methods that allow to automatically identify fraudulent transactions. Covering various aspects, such as processing large amounts of data, integrating with blockchain networks, and ensuring data security, is key to developing this system.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Машинне навчання – підрозділ штучного інтелекту, що дозволяє комп'ютерам навчатися на основі даних без явного програмування;

Нейронна мережа – обчислювальна модель, натхненна біологічними нейронними мережами, що використовується для розпізнавання патернів та класифікації;

Блокчейн – розподілена база даних, яка підтримує список впорядкованих записів, що називаються блоками, захищених криптографічними методами;

Транзакція – операція передачі криптовалюти або іншого активу між учасниками блокчейн-мережі;

Шахрайська транзакція – транзакція, що здійснюється з метою обману, викрадення коштів або інших шахрайських дій;

API – Application Programming Interface – інтерфейс програмування додатків;

CNN – Convolutional Neural Network – конволюційна нейронна мережа;

DB – Database – база даних;

LSTM – Long Short-Term Memory – довготривала короткострокова пам'ять;

ML – Machine Learning – машинне навчання;

NoSQL – Not Only SQL – не тільки SQL;

RNN – Recurrent Neural Network – рекурентна нейронна мережа;

SQL – Structured Query Language – структурована мова запитів.

ВСТУП

Станом на 2024 рік, питання безпеки блокчейн-транзакцій стає все більш актуальним, оскільки блокчейн-технології знаходять застосування у різних сферах, від фінансових операцій до зберігання даних. Незважаючи на те, що блокчейн забезпечує високий рівень безпеки завдяки своїй децентралізованій природі, шахрайські транзакції залишаються серйозною загрозою. Існуючі методи виявлення шахрайства в блокчейні часто обмежені та не забезпечують достатньо високої точності та швидкості виявлення.

Для автоматизації процесу виявлення шахрайських транзакцій в блокчейн-мережі актуально розробити систему, яка використовує алгоритми машинного навчання. Це дозволить покращити ефективність виявлення шахрайства та знизити фінансові втрати, пов'язані з шахрайськими діями.

Об’єкт розробки – процес виявлення шахрайських транзакцій у блокчейн-мережі. Предметом розробки є алгоритми класифікації транзакцій у блокчейн-мережі як шахрайських або не шахрайських.

Мета роботи – розробка системи для автоматичного виявлення шахрайських транзакцій у блокчейні.

Оцінка сучасного стану проблеми – на сучасному етапі розвитку блокчейн-технологій, існуючі методи виявлення шахрайства залишаються недостатньо ефективними. Аналіз існуючих рішень показує необхідність використання новітніх технологій машинного навчання для покращення точності та швидкості виявлення шахрайських транзакцій. Існуючі системи часто не враховують всіх аспектів транзакцій, що знижує їхню ефективність.

Актуальність роботи та підстави для її виконання – сучасний ритм розвитку цифрових технологій та збільшення обсягів фінансових операцій у блокчейні визначають високий попит на засоби для виявлення шахрайства. Однак існуючі рішення не завжди відповідають реальним

потребам користувачів. Розробка системи для автоматичного виявлення шахрайських транзакцій у блокчейні є актуальною як з погляду технологічних можливостей, так і з точки зору забезпечення безпеки фінансових операцій.

Мета й завдання дослідження – головною метою цього дослідження є розробка системи, яка забезпечить автоматичне виявлення шахрайських транзакцій у блокчейні. Система має стати важливим інструментом для підвищення безпеки фінансових операцій. Завдання включають розробку алгоритмів обробки даних, інтеграцію з блокчейн-платформами та забезпечення максимальної точності виявлення шахрайства.

Об’єкт дослідження та предмет дослідження – об'єктом дослідження є процес виявлення шахрайських транзакцій у блокчейн-мережі, а предметом дослідження є алгоритми машинного навчання та методи класифікації, які дозволяють автоматично ідентифікувати шахрайські транзакції. Охоплення різних аспектів, таких як обробка великих обсягів даних, інтеграція з блокчейн-мережами та забезпечення безпеки даних, є ключовим для розробки цієї системи.

Методи дослідження – аналіз літератури, аналіз існуючих технологій для виявлення шахрайства, дослідження методів машинного навчання, моделювання системи, концептуальне та фізичне проектування бази даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Блокчейн є інноваційною технологією, яка забезпечує прозорість, безпеку і децентралізацію транзакцій. Основна ідея блокчейну полягає у створенні незмінного реєстру транзакцій, що зберігаються у вигляді ланцюжка блоків, де кожен блок містить хеш попереднього блоку, тим самим забезпечуючи цілісність і незмінність даних. Блокчейн може застосовуватися у різних сферах, таких як фінанси, логістика, охорона здоров'я, енергетика та інші, забезпечуючи безпечний обмін інформацією та транзакціями.

Блокчейн складається з мережі вузлів (ноди), кожен з яких зберігає копію повного реєстру транзакцій. Кожна транзакція, після її підтвердження, додається до блоку, який потім додається до ланцюжка блоків. Цей процес називається майнінгом і забезпечує цілісність та незмінність даних у блокчейні. Майнери, які виконують обчислювальні завдання для додавання нових блоків, отримують винагороду у вигляді криптовалюти.

Основними перевагами блокчейн-технологій є децентралізація, прозорість, безпека та незмінність даних. Децентралізація означає, що немає центрального органу, який контролює мережу, що робить її стійкою до зломів та маніпуляцій. Прозорість забезпечується відкритим доступом до всієї історії транзакцій, що дозволяє будь-кому перевірити коректність операцій. Безпека досягається через використання криптографічних методів, таких як хешування та цифрові підписи, які забезпечують цілісність та аутентичність даних.

Шахрайські транзакції у блокчейні можуть включати різні види атак, такі як подвійне витрачання (double-spending), фішинг, атаки на смарт-контракти, підроблені ICO (Initial Coin Offering) та інші види шахрайства.

Подвійне витрачання полягає у тому, що зловмисник намагається витратити одні й ті самі криптовалюти більше ніж один раз. Фішинг-атаки націлені на отримання доступу до особистих даних користувачів, що дозволяє зловмисникам отримувати контроль над їхніми криптовалютними гаманцями. Атаки на смарт-контракти використовують вразливості у коді для крадіжки активів або маніпуляції з даними.

Феномен подвійного витрачання є однією з основних загроз для блокчейну, оскільки він підриває основну ідею незмінності транзакцій. У разі успішної атаки з подвійним витрачанням зловмисник може витратити ті ж самі криптовалюти більше ніж один раз, що призводить до фінансових втрат для інших учасників мережі. Для запобігання таким атакам використовуються механізми консенсусу, такі як Proof of Work (PoW) або Proof of Stake (PoS), які ускладнюють зміну блоків після їх додавання до ланцюжка.

Фішинг є ще однією поширеною формою шахрайства у блокчейн-мережах. Зловмисники створюють фальшиві веб-сайти або надсилають електронні листи, що імітують легітимні сервіси, з метою отримання конфіденційної інформації від користувачів, такої як приватні ключі або паролі. Отримавши ці дані, шахраї можуть отримати доступ до криптовалютних гаманців користувачів і вкрасти їхні активи. Захист від фішингу включає підвищення обізнаності користувачів про безпеку, використання двофакторної аутентифікації та регулярні перевірки безпеки.

Атаки на смарт-контракти є ще однією значною загрозою для блокчейн-мереж. Смарт-контракти – це програмні коди, що автоматизують виконання угод на блокчейні. Вони можуть бути використані для автоматизації фінансових операцій, управління активами, укладення контрактів тощо. Однак, якщо смарт-контракт містить вразливості, зловмисники можуть використати їх для маніпуляцій або крадіжки активів. Наприклад, атака на смарт-контракт DAO у 2016 році призвела до втрати мільйонів доларів у криптовалюті Ethereum.

Важливим аспектом безпеки блокчейн-мереж є захист від підроблених ICO. ICO (Initial Coin Offering) – це спосіб залучення інвестицій для нових блокчейн-проектів через продаж токенів. Однак, деякі зловмисники створюють підроблені ICO, збираючи гроші від інвесторів, але не маючи наміру реалізовувати проект. Це призводить до значних фінансових втрат для інвесторів та підриває довіру до блокчейн-спільноти.

Для виявлення та запобігання шахрайським транзакціям у блокчейні використовуються різні методи та технології. Одним з таких методів є використання алгоритмів машинного навчання для аналізу транзакційних даних та виявлення підозрілих патернів. Наприклад, нейронні мережі можуть бути навчені на великих обсягах даних для виявлення аномальних транзакцій, що можуть свідчити про шахрайство. Ці алгоритми можуть автоматично аналізувати тисячі транзакцій у реальному часі, виявляючи підозрілі активності.

Машинне навчання також може використовуватися для класифікації транзакцій як легітимних або шахрайських. Для цього використовуються різні моделі, такі як логістична регресія, дерева рішень, випадкові ліси (random forests) та глибокі нейронні мережі. Ці моделі можуть враховувати різні ознаки транзакцій, такі як сума, частота, час проведення, а також історію транзакцій користувача, для прийняття рішення про ймовірність шахрайства.

Однією з основних переваг використання машинного навчання для виявлення шахрайських транзакцій є здатність адаптуватися до нових видів шахрайства. Оскільки шахраї постійно змінюють свої методи, традиційні правила та фільтри можуть бути недостатньо ефективними. Машинне навчання дозволяє системі самостійно навчатися на нових даних та виявляти нові патерни шахрайства, що забезпечує більш ефективний захист.

Крім машинного навчання, для виявлення шахрайських транзакцій у блокчейні використовуються методи аналізу графів. Блокчейн може бути представлений у вигляді графу, де вузли представляють адреси або

користувачів, а ребра – транзакції між ними. Аналіз графів дозволяє виявляти підозрілі зв'язки між вузлами, такі як високий ступінь зв'язності або циклічні структури, що можуть свідчити про шахрайську активність.

Алгоритми аналізу графів, такі як алгоритми виявлення спільнот, можуть використовуватися для виявлення груп вузлів, які взаємодіють між собою більше, ніж з іншими вузлами. Це може допомогти виявити злочинні мережі або шахрайські схеми, де кілька адрес використовуються для приховування джерела коштів або проведення підозрілих транзакцій.

Іншим підходом до виявлення шахрайських транзакцій є використання методів аналізу поведінкових патернів. Це включає аналіз звичок та поведінки користувачів, таких як частота та час проведення транзакцій, типи адрес, з якими вони взаємодіють, і суми переказів. Відхилення від нормальної поведінки можуть свідчити про шахрайську активність.

Окрім технічних методів, важливою частиною боротьби з шахрайством у блокчейні є підвищення обізнаності користувачів. Багато шахрайських схем, таких як фішинг, ґрунтуються на недостатній поінформованості користувачів про ризики та способи захисту. Проведення освітніх кампаній, публікація порад щодо безпеки та регулярні оновлення програмного забезпечення можуть значно знизити ризики шахрайства.

Таким чином, блокчейн є потужною технологією, що забезпечує безпечний та прозорий обмін інформацією і транзакціями. Однак, шахрайські транзакції залишаються значною загрозою, що потребує постійної уваги та розвитку нових методів захисту. Використання машинного навчання, аналізу графів та поведінкових патернів, а також підвищення обізнаності користувачів є ключовими елементами ефективної боротьби з шахрайством у блокчейні.

1.2 Аналіз існуючих технологій

Транзакції у блокчейні є незмінними записами про передачу активів між користувачами. Кожна транзакція має унікальний ідентифікатор та містить інформацію про відправника, одержувача, суму та часову мітку. Ця інформація є основою для аналізу та класифікації транзакцій. Існуючі технології для виявлення шахрайських транзакцій використовують різні підходи, зокрема методи машинного навчання, аналіз графів та поведінкові моделі.

Однією з основних технологій, що застосовується для виявлення шахрайських транзакцій, є машинне навчання. Машинне навчання дозволяє системам автоматично навчатися на основі даних і покращувати свою продуктивність без явного програмування. Однією з популярних моделей машинного навчання є логістична регресія. Логістична регресія використовується для бінарної класифікації, тобто для визначення ймовірності того, що транзакція є шахрайською або ні.

Формула для логістичної регресії виглядає наступним чином:

$$\mathrm{P}(Y = 1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_n X_n)}} \quad (1.1)$$

де $\mathrm{P}(Y = 1|X)$ – ймовірність того, що транзакція є шахрайською;

β_0 – вільний член;

$\beta_1 + \beta_2 + \cdots + \beta_n$ – коефіцієнти регресії;

$X_1 + X_2 + \cdots + X_n$ – ознаки транзакції.

Іншою популярною моделлю є дерева рішень. Дерева рішень використовують структуру дерева для прийняття рішень на основі ознак транзакцій. Кожен вузол дерева представляє ознаку, кожна гілка – умову на цю ознаку, а кожен лист – клас (шахрайська або не шахрайська транзакція). Процес побудови дерева рішень включає вибір найбільш значущих ознак і створення умов, що мінімізують кількість помилок класифікації.

Приклад дерева рішень для виявлення шахрайських транзакцій представлений на рисунку 1.1.

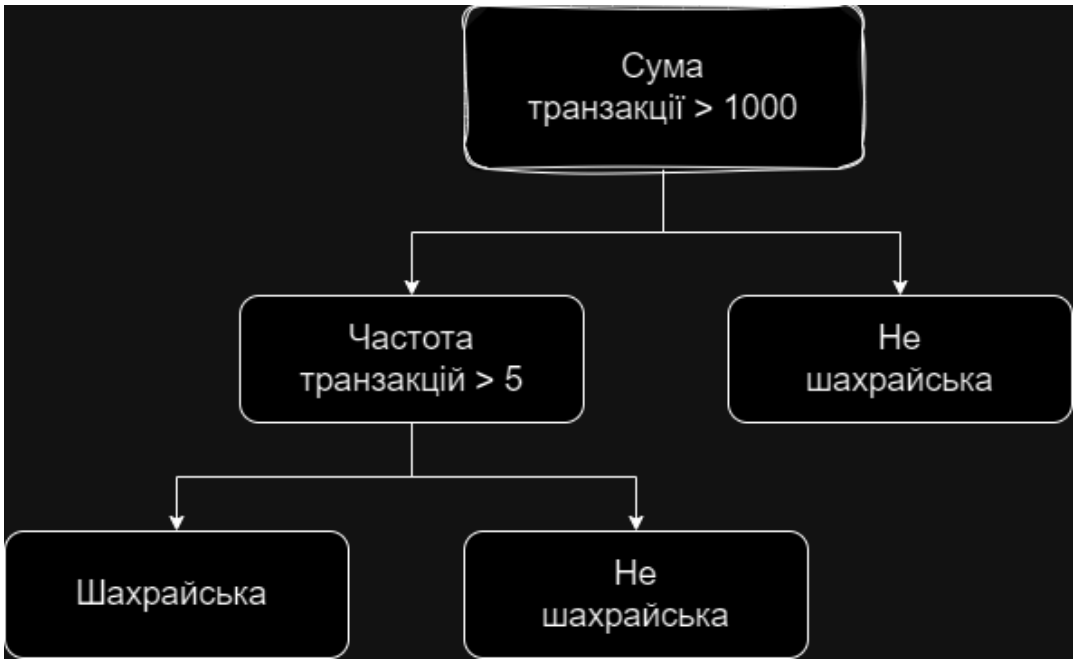

Рисунок 1.1 – Приклад дерева рішень для виявлення шахрайських транзакцій

Ще одним потужним методом є випадкові ліси (Random Forests). Випадкові ліси складаються з множини дерев рішень, кожне з яких будується на випадковій підмножині даних та ознак. Кінцеве рішення приймається шляхом голосування всіх дерев. Цей метод знижує ризик перенавчання та підвищує точність класифікації.

Формула для обчислення прогнозу у випадковому лісі виглядає так:

$$\check{y} = \frac{1}{M}\sum_{m=1}^{M} h_m(x) \tag{1.2}$$

де M – кількість дерев у лісі;

$h_m(x)$ – прогноз m-го дерева для прикладу x.

Глибоке навчання є ще однією передовою технологією для виявлення шахрайських транзакцій. Глибоке навчання використовує багатошарові нейронні мережі для автоматичного вилучення ознак та класифікації даних. Однією з популярних архітектур для аналізу часових рядів є рекурентні нейронні мережі (RNN), зокрема Long Short-Term Memory (LSTM).

LSTM здатні зберігати та оновлювати інформацію протягом довгих періодів часу, що робить їх ефективними для аналізу послідовних даних, таких як транзакції у блокчейні. Архітектура LSTM включає в себе комірки пам'яті, ворота забування, ворота введення та ворота виходу, що дозволяє моделі контролювати потік інформації.

Формули для LSTM виглядають наступним чином:

– ворота забування:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f); \quad (1.3)$$

– ворота введення:

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i); \quad (1.4)$$

– оновлення стану пам'яті:

$$\widetilde{C_t} = \tanh(W_C * [h_{t-1}, x_t] + b_C); \quad (1.5)$$

– новий стан пам'яті:

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}; \quad (1.6)$$

– ворота виходу:

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o); \quad (1.7)$$

– новий вихід:

$$h_t = o_t * \tanh(C_t). \quad (1.8)$$

Конволюційні нейронні мережі (CNN) також можуть бути використані для виявлення шахрайських транзакцій. CNN здатні автоматично вилучати складні ознаки з даних і можуть бути ефективними для аналізу структурованих даних, таких як графи транзакцій.

Приклад архітектури CNN для виявлення шахрайських транзакцій може включати кілька конволюційних шарів, шарів підсумування (pooling), повнозв'язних шарів та вихідний шар з активацією softmax для класифікації.

Аналіз графів є ще одним важливим підходом для виявлення шахрайських транзакцій. Блокчейн може бути представлений у вигляді графу, де вузли представляють адреси або користувачів, а ребра – транзакції між ними. Аналіз графів дозволяє виявляти підозрілі зв'язки між вузлами, такі як високий ступінь зв'язності або циклічні структури, що можуть свідчити про шахрайську активність.

Алгоритми аналізу графів, такі як алгоритми виявлення спільнот, можуть використовуватися для виявлення груп вузлів, які взаємодіють між собою більше, ніж з іншими вузлами. Це може допомогти виявити злочинні мережі або шахрайські схеми, де кілька адрес використовуються для приховування джерела коштів або проведення підозрілих транзакцій.

Наприклад, алгоритм виявлення спільнот Louvain дозволяє ідентифікувати спільноти у великих графах за допомогою ітеративного процесу, що максимізує модульність. Модульність – це міра, яка визначає якість розподілу графу на спільноти:

$$Q = \frac{1}{2m}\sum_{i,j}[A_{ij} - \frac{k_i k_j}{2m}]\delta(c_i, c_j), \quad (1.9)$$

де A_{ij} – елемент матриці суміжності;

k_i та k_j – ступені вузлів і та j;

m – кількість ребер у графі;

c_i та c_j – спільноти, до яких належать вузли і та j;

$\delta(c_i, c_j)$ – функція Кронекера, що дорівнює 1, якщо $c_i = c_j$, і 0 в іншому випадку.

Аналіз графів також може включати виявлення центральних вузлів, які мають високий вплив на мережу. Алгоритми центральності, такі як центральність за ступенем, проміжна центральність (betweenness centrality) та власний вектор (eigenvector centrality), дозволяють ідентифікувати вузли, які відіграють ключову роль у розповсюдженні інформації або коштів у мережі.

Наприклад, проміжна центральність вимірює, наскільки часто вузол знаходиться на найкоротших шляхах між іншими вузлами:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{1.10}$$

де σ_{st} – кількість найкоротших шляхів між вузлами s та t;

$\sigma_{st}(v)$ – кількість найкоротших шляхів між вузлами s та t, що проходять через вузол (v).

Крім методів машинного навчання та аналізу графів, для виявлення шахрайських транзакцій використовуються поведінкові моделі. Поведінкові моделі аналізують звички та поведінку користувачів, такі як частота та час проведення транзакцій, типи адрес, з якими вони взаємодіють, і суми переказів. Відхилення від нормальної поведінки можуть свідчити про шахрайську активність.

Наприклад, якщо користувач зазвичай проводить невеликі транзакції у певний час доби, а раптом починає проводити великі транзакції у незвичний час, це може бути ознакою компрометації акаунта. Аналіз поведінкових патернів може використовувати статистичні методи, такі як

коефіцієнт відхилення, або методи машинного навчання, такі як кластеризація та аномалійне виявлення.

Кластеризація є одним з методів, що використовується для виявлення груп схожих транзакцій або користувачів. Один з популярних алгоритмів кластеризації – K-середніх (K-means). K-means намагається розбити дані на k кластерів, мінімізуючи відстань між точками всередині кожного кластера та їхнім центроїдом.

Формула для оновлення центроїдів у алгоритмі K-means:

$$\mu_i = \frac{1}{|C_i|}\sum_{x_j \in C_i} x_j, \tag{1.11}$$

де μ_i – центроїд і-го кластера;

C_i – набір точок у і-му кластері;

x_j – точка даних.

Кластеризація дозволяє виявляти аномалії як точки, що знаходяться далеко від центроїдів своїх кластерів. Аномалійне виявлення також може використовувати такі методи, як метод відокремлення (Isolation Forest) або метод підтримкових векторів (One-Class SVM).

Метод відокремлення (Isolation Forest) базується на ідеї, що аномальні точки знаходяться на меншій глибині у дереві, що відокремлює дані. Алгоритм будує ансамбль дерев відокремлення та використовує середню глибину для визначення аномалій.

Формула для обчислення аномалійного оцінювання у методі відокремлення:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{1.12}$$

де $s(x, n)$ – аномалійне оцінювання точки x у наборі даних з n точок;

E(h(x)) – середня глибина точки x у дереві відокремлення;

c(n) – середня глибина випадкового дерева відокремлення.

Метод підтримкових векторів (One-Class SVM) також використовується для аномалійного виявлення. One-Class SVM намагається знайти гіперплощину, що відокремлює більшість даних від аномалій.

Формула для One-Class SVM:

$$min_{w,p} \frac{1}{2}\|w\|^2 + \frac{1}{vn}\sum_{i=1}^{n} \max(0, p - w * x_i) - p, \qquad (1.13)$$

де w – вектор ваг;

p – відступ (offset);

v – параметр регуляризації;

x_i – точка даних.

Таким чином, існує безліч технологій та методів, що використовуються для виявлення шахрайських транзакцій у блокчейні. Кожен з них має свої переваги та недоліки, і вибір методу залежить від специфіки завдання та доступних даних. Комбінація різних методів, таких як машинне навчання, аналіз графів та поведінкові моделі, може забезпечити більш точне та ефективне виявлення шахрайства.

Конкретні приклади існуючих рішень:

– Chainalysis: ця компанія використовує аналітику великих даних для моніторингу та виявлення підозрілих транзакцій. Chainalysis пропонує інструменти для відстеження походження криптовалют, виявлення підозрілих адрес і транзакцій, а також створення звітів для правоохоронних органів;

– Elliptic: цей сервіс забезпечує моніторинг транзакцій у реальному часі для виявлення підозрілої активності. Elliptic використовує алгоритми машинного навчання для аналізу транзакційних даних та ідентифікації аномальних патернів, які можуть свідчити про шахрайство.

Недоліки існуючих технологій:

– висока складність алгоритмів: багато методів машинного навчання потребують значних обчислювальних ресурсів та складних алгоритмів для аналізу даних;

– потреба у великих обсягах даних: для ефективного навчання моделей необхідно мати великі обсяги даних, що можуть бути недоступними для деяких організацій;

– обчислювальні ресурси: використання глибоких нейронних мереж потребує значних обчислювальних ресурсів, таких як графічні процесори (GPU).

Для покращення ефективності виявлення шахрайських транзакцій можна використовувати гібридні методи, що поєднують кілька різних алгоритмів, а також розробляти спеціалізовані алгоритми для аналізу блокчейн-транзакцій.

1.3 Сфера використання

Системи для виявлення шахрайських транзакцій у блокчейні мають широке застосування у різних сферах, де безпека та цілісність фінансових операцій є критично важливими. Основні сфери використання таких систем включають фінансові установи, біржі криптовалют, державні органи, комерційні підприємства, постачальники послуг, та інші. Детальний аналіз кожної сфери допоможе зрозуміти, як саме ці технології можуть бути впроваджені для забезпечення безпеки та прозорості транзакцій.

Фінансові установи.

Банки та інші фінансові установи є одними з основних користувачів систем для виявлення шахрайських транзакцій. Вони використовують блокчейн для проведення міжнародних переказів, управління активами, кредитування та інших фінансових операцій. Виявлення шахрайських транзакцій допомагає знизити ризики фінансових втрат, захистити клієнтів від шахрайства та підвищити довіру до фінансової системи. Наприклад,

впровадження систем моніторингу на основі машинного навчання дозволяє банкам автоматично виявляти підозрілі операції та оперативно реагувати на них.

Біржі криптовалют.

Біржі криптовалют також є важливими користувачами систем для виявлення шахрайських транзакцій. Біржі здійснюють великий обсяг транзакцій щодня, що робить їх привабливою мішенню для зловмисників. Системи моніторингу транзакцій допомагають біржам виявляти та блокувати підозрілі операції, захищати користувачів від фішингу, подвійного витрачання та інших видів шахрайства. Наприклад, біржі можуть використовувати алгоритми аналізу графів для виявлення шахрайських схем та зловмисних адрес.

Державні органи.

Регуляторні та правоохоронні органи можуть використовувати системи для моніторингу транзакцій з метою боротьби з відмиванням грошей, фінансуванням тероризму та іншими незаконними діями. Вони можуть відстежувати підозрілі фінансові операції, виявляти злочинні мережі та проводити розслідування на основі аналізу блокчейн-даних. Наприклад, використання технологій машинного навчання дозволяє державним органам автоматично ідентифікувати підозрілі транзакції та генерувати звіти для подальшого розслідування.

Комерційні підприємства.

Великі комерційні підприємства, що використовують блокчейн для управління ланцюгами постачання, можуть використовувати системи для виявлення шахрайських транзакцій для забезпечення прозорості та безпеки своїх операцій. Виявлення підробок, незаконних транзакцій та інших видів шахрайства дозволяє підприємствам знизити ризики втрат, забезпечити відповідність нормативним вимогам та підвищити довіру до своїх продуктів та послуг. Наприклад, компанії можуть використовувати блокчейн для

відстеження руху товарів по ланцюгу постачання та виявлення підозрілих транзакцій на будь-якому етапі.

Постачальники послуг.

Постачальники фінансових та технологічних послуг, такі як платіжні системи, цифрові гаманці та інші, також можуть використовувати системи для виявлення шахрайських транзакцій для захисту своїх клієнтів. Впровадження таких систем дозволяє автоматично аналізувати транзакції, виявляти аномалії та запобігати шахрайським операціям. Наприклад, платіжні системи можуть використовувати нейронні мережі для аналізу поведінки користувачів та виявлення підозрілих патернів.

Освітні та науково-дослідні установи.

Університети та науково-дослідні інститути можуть використовувати системи для виявлення шахрайських транзакцій у своїх дослідженнях. Вивчення ефективності різних методів виявлення шахрайства, розробка нових алгоритмів та аналіз результатів можуть допомогти покращити існуючі технології та створити нові підходи до захисту блокчейн-мереж. Наприклад, дослідники можуть проводити експерименти з використанням різних моделей машинного навчання для виявлення шахрайських транзакцій та порівнювати їх ефективність.

Благодійні організації.

Благодійні організації можуть використовувати блокчейн для забезпечення прозорості своїх фінансових операцій та підвищення довіри з боку донорів. Системи для виявлення шахрайських транзакцій допомагають захистити благодійні фонди від незаконних дій та забезпечити правильне використання коштів. Наприклад, використання блокчейн дозволяє відстежувати рух коштів від донорів до кінцевих отримувачів та виявляти підозрілі транзакції.

Міжнародні організації та проекти.

Міжнародні організації, такі як ООН, Світовий банк та інші, можуть використовувати блокчейн для управління міжнародними проектами та

фінансовими транзакціями. Впровадження систем для виявлення шахрайських транзакцій допомагає забезпечити прозорість та підзвітність у використанні фінансових ресурсів. Наприклад, блокчейн може бути використаний для моніторингу фінансування міжнародних гуманітарних програм та виявлення шахрайства.

Розробники програмного забезпечення.

Розробники програмного забезпечення можуть використовувати технології для виявлення шахрайських транзакцій при створенні додатків та платформ на основі блокчейну. Це дозволяє інтегрувати функції безпеки безпосередньо у продукти та забезпечити захист користувачів від шахрайства. Наприклад, розробники можуть впроваджувати алгоритми машинного навчання у криптовалютні гаманці для автоматичного аналізу транзакцій та виявлення аномалій.

Користувачі криптовалют.

Індивідуальні користувачі криптовалют також можуть отримати вигоду від систем для виявлення шахрайських транзакцій. Використання таких систем дозволяє підвищити безпеку власних активів, уникнути шахрайських схем та забезпечити прозорість своїх фінансових операцій. Наприклад, користувачі можуть використовувати додатки, що аналізують їхні транзакції та повідомляють про підозрілу активність.

Таким чином, сфера використання систем для виявлення шахрайських транзакцій у блокчейні є надзвичайно широкою і охоплює різні галузі. Впровадження таких систем допомагає забезпечити безпеку, прозорість та довіру до фінансових операцій, знижуючи ризики шахрайства та фінансових втрат. Кожна з наведених сфер може виграти від використання передових технологій для аналізу та класифікації транзакцій, забезпечуючи ефективний захист від зловмисних дій.

1.4 Постановка задачі

Для розробки ефективної системи виявлення шахрайських транзакцій у блокчейні необхідно чітко визначити завдання, які потрібно виконати. Постановка задачі включає в себе визначення основних етапів розробки, встановлення вимог до системи, вибір методів та інструментів, а також планування процесу реалізації та тестування. Детальний аналіз кожного етапу допоможе забезпечити комплексний підхід до вирішення проблеми та досягнення високої ефективності системи.

Перший етап розробки системи полягає у зборі та підготовці даних про транзакції з блокчейн-мережі. Необхідно зібрати історичні дані про транзакції, включаючи інформацію про відправника, одержувача, суму, часову мітку та інші атрибути. Дані можуть бути зібрані з публічних блокчейн-мереж, таких як Bitcoin або Ethereum, або з приватних блокчейнів, залежно від специфіки завдання. Окрім збору даних, важливо провести їх очищення та нормалізацію, видаляючи дублікатів, обробляючи відсутні значення та масштабуючи числові ознаки для подальшого аналізу.

На другому етапі необхідно визначити, які методи машинного навчання та глибокого навчання будуть використані для виявлення шахрайських транзакцій. Вибір методів залежить від специфіки даних, обсягу доступної інформації та вимог до точності та продуктивності системи. Популярні методи включають логістичну регресію, дерева рішень, випадкові ліси, рекурентні нейронні мережі (LSTM) та конволюційні нейронні мережі (CNN). Кожен з методів має свої переваги та недоліки, тому можливо знадобиться використання кількох методів та їх порівняння для досягнення оптимальних результатів.

Для ефективного виявлення шахрайських транзакцій необхідно розробити архітектуру нейронної мережі, яка буде здатна аналізувати великі обсяги даних та виявляти складні патерни. Архітектура може включати кілька шарів, таких як вхідний шар, приховані шари та вихідний шар.

Приховані шари можуть включати LSTM- або CNN-шари, залежно від типу даних. Важливо також визначити гіперпараметри моделі, такі як кількість нейронів у кожному шарі, швидкість навчання та кількість епох навчання.

Після розробки архітектури необхідно провести навчання моделі на зібраних даних. Для цього дані діляться на тренувальний, валідаційний та тестовий набори. Тренувальний набір використовується для навчання моделі, валідаційний набір для налаштування гіперпараметрів, а тестовий набір для оцінки ефективності моделі. Під час навчання використовуються методи оптимізації, такі як стохастичний градієнтний спуск (SGD) або Adam, для мінімізації функції втрат та покращення точності класифікації. Результати навчання оцінюються за допомогою метрик, таких як точність, повнота, специфічність та F1-міра.

Після навчання та тестування моделі необхідно інтегрувати її у блокчейн-систему для моніторингу транзакцій у реальному часі. Для цього розробляється API, через яке інші сервіси зможуть взаємодіяти з системою. API забезпечує отримання даних про нові транзакції, передачу їх до моделі для аналізу та повернення результатів класифікації. Інтеграція може включати також розробку інтерфейсу користувача для моніторингу та керування системою.

На наступному етапі проводиться валідація системи для перевірки її ефективності у реальних умовах. Це включає тестування системи на нових даних, оцінку результатів класифікації та виявлення можливих проблем. Після валідації проводиться оптимізація моделі для покращення її продуктивності та точності. Оптимізація може включати налаштування гіперпараметрів, використання додаткових даних для навчання та вдосконалення архітектури нейронної мережі.

Для забезпечення ефективного моніторингу та реагування на шахрайські транзакції необхідно розробити механізми звітування та нотифікацій. Система повинна автоматично генерувати звіти про виявлені підозрілі транзакції та надсилати нотифікації відповідним користувачам або

адміністраторам. Це дозволяє оперативно реагувати на загрози та запобігати фінансовим втратам. Звіти можуть включати детальну інформацію про транзакції, причини їх класифікації як шахрайських та рекомендації щодо подальших дій.

Тестування є важливим етапом розробки, який забезпечує надійність та коректність роботи системи. Воно включає юніт-тести для перевірки окремих компонентів, інтеграційні тести для перевірки взаємодії між компонентами та системні тести для оцінки роботи системи в цілому. Тестування повинно охоплювати різні сценарії, включаючи нормальні умови роботи, високі навантаження та аварійні ситуації. Результати тестування допомагають виявити та виправити помилки, покращити продуктивність та забезпечити стабільну роботу системи.

Для забезпечення ефективного використання системи необхідно розробити детальну документацію, яка включатиме інструкції з установки, налаштування та використання системи, а також технічну документацію для розробників. Окрім цього, важливо провести навчання користувачів, яке може включати семінари, тренінги та онлайн-курси. Це допоможе користувачам зрозуміти, як правильно використовувати систему, які функції вона надає та як реагувати на виявлені шахрайські транзакції.

Після запуску системи необхідно забезпечити її підтримку та регулярне оновлення. Це включає моніторинг роботи системи, виявлення та виправлення помилок, додавання нових функцій та вдосконалення існуючих алгоритмів. Підтримка також може включати реагування на запити користувачів, проведення регулярних аудитів безпеки та оновлення програмного забезпечення для забезпечення відповідності сучасним вимогам та загрозам.

2 ВИЯВЛЕННЯ ВИМОГ ДО РОЗРОБЛЕНОЇ ТЕХНОЛОГІЇ

2.1 Системні вимоги

Системні вимоги є ключовим аспектом розробки будь-якої програмної системи, оскільки вони визначають апаратні та програмні ресурси, необхідні для її функціонування. Для системи виявлення шахрайських транзакцій у блокчейні, яка повинна обробляти великі обсяги даних та здійснювати складні обчислення у реальному часі, важливо врахувати як апаратні, так і програмні вимоги, що забезпечать її ефективність та надійність.

Система повинна працювати на операційних системах, які підтримують обробку великих обсягів даних та мають високий рівень безпеки. Найпоширенішими операційними системами для таких завдань є Linux та Windows. Linux, зокрема, має перевагу завдяки своїй стабільності, масштабованості та широкому спектру інструментів для аналізу даних та розробки програмного забезпечення. Також можлива підтримка macOS, особливо для розробників, що використовують цю платформу.

Оскільки система потребує обробки великого обсягу даних у реальному часі, необхідно забезпечити достатні обчислювальні ресурси. Для цього рекомендується використовувати потужні процесори з багатоядерною архітектурою, такі як Intel Xeon або AMD EPYC. Мінімальні вимоги до центрального процесора (CPU) включають підтримку технологій, таких як SSE4.2 для оптимізації обчислень. Рекомендується мати не менше 16 GB оперативної пам'яті (RAM) для забезпечення ефективної обробки даних та запобігання затримкам у роботі системи.

Для прискорення обчислень та навчання моделей глибокого навчання рекомендується використовувати графічні процесори (GPU). Найбільш підходящими є GPU від компаній NVIDIA (серії Tesla, Quadro, GeForce) або AMD (серії Radeon, Instinct). GPU забезпечують значне прискорення

обчислень порівняно з CPU, що є критично важливим для задач, пов'язаних з машинним та глибоким навчанням.

Важливим аспектом є також наявність швидких накопичувачів даних. Рекомендується використовувати твердотільні накопичувачі (SSD) замість традиційних жорстких дисків (HDD) для забезпечення швидкого доступу до даних та прискорення процесів читання/запису. Обсяг накопичувача залежить від кількості даних, які потрібно обробляти, але мінімальні вимоги складають 1 TB SSD.

Для забезпечення швидкого та надійного доступу до блокчейн-мережі необхідно мати високошвидкісне мережеве обладнання. Рекомендується використовувати мережеві інтерфейси з підтримкою швидкості передачі даних не менше 1 Gbps. Це дозволить забезпечити оперативне оновлення даних транзакцій та швидке реагування на виявлені аномалії.

Важливо визначити, на якій блокчейн-платформі буде працювати система. Найпоширенішими платформами є Ethereum, Bitcoin, Hyperledger та інші. Вибір платформи залежить від вимог проекту та специфіки завдань. Наприклад, для проектів з використанням смарт-контрактів може бути доцільно обрати Ethereum, а для корпоративних рішень – Hyperledger.

Для зберігання та обробки транзакційних даних необхідно використовувати потужні та масштабовані бази даних. Рекомендується використовувати реляційні бази даних, такі як PostgreSQL або MySQL, для зберігання структурованих даних. Для обробки великих обсягів неструктурованих даних та швидкого пошуку підходять NoSQL бази даних, такі як MongoDB або Cassandra.

Для розробки системи необхідно вибрати мови програмування, які забезпечують високу продуктивність та підтримують роботу з великими даними та машинним навчанням. Найпопулярнішою мовою для таких завдань є Python, завдяки широкому спектру бібліотек для машинного навчання та аналізу даних, таких як TensorFlow, PyTorch, Pandas та NumPy. Також можуть використовуватися інші мови, такі як R для статистичного

аналізу, C++ для високопродуктивних обчислень та Java для інтеграції з корпоративними системами.

Важливо вибрати відповідні фреймворки для реалізації моделей машинного та глибокого навчання. Найпопулярнішими фреймворками є TensorFlow та PyTorch, які забезпечують широкі можливості для створення, навчання та розгортання нейронних мереж. Інші корисні інструменти включають Scikit-learn для класичних алгоритмів машинного навчання, Keras як високорівневий API для TensorFlow та Dask для паралельної обробки даних.

Для забезпечення масштабованості та керованості системи необхідно використовувати інструменти для оркестрації та розгортання контейнерів, такі як Docker та Kubernetes. Docker дозволяє контейнеризувати додатки, забезпечуючи їхню портативність та ізоляцію. Kubernetes забезпечує автоматичне масштабування, балансування навантаження та управління розгортанням контейнерів, що є критично важливим для систем з високими вимогами до продуктивності та надійності.

Для забезпечення стабільної роботи системи необхідно впровадити інструменти для моніторингу та логування. Наприклад, Prometheus може використовуватися для збору та аналізу метрик продуктивності, а Grafana для візуалізації даних. Для логування подій та виявлення помилок можна використовувати ELK Stack (Elasticsearch, Logstash, Kibana), що дозволяє зберігати та аналізувати лог-файли у реальному часі.

Безпека є критично важливим аспектом для системи виявлення шахрайських транзакцій. Необхідно забезпечити захист даних та системи від несанкціонованого доступу, використовуючи методи шифрування, аутентифікації та авторизації. Також важливо забезпечити регулярні перевірки безпеки, оновлення програмного забезпечення та моніторинг активності для виявлення потенційних загроз.

Для запобігання втратам даних та забезпечення безперервності роботи системи необхідно впровадити механізми резервного копіювання та

відновлення. Це включає регулярне створення резервних копій баз даних та конфігурацій системи, а також тестування процедур відновлення для забезпечення їхньої ефективності.

Таким чином, системні вимоги для розробки системи виявлення шахрайських транзакцій у блокчейні охоплюють широкий спектр апаратних та програмних ресурсів. Врахування всіх аспектів, від обчислювальних ресурсів до безпеки та резервного копіювання, допоможе забезпечити ефективну та надійну роботу системи, що є критично важливим для захисту фінансових операцій та запобігання шахрайству.

2.2 Функціональні вимоги

Функціональні вимоги визначають, які конкретні функції та завдання повинна виконувати система для виявлення шахрайських транзакцій у блокчейні. Вони охоплюють весь спектр можливостей системи, від збору даних до виявлення та реагування на шахрайські транзакції. Чітке визначення функціональних вимог є ключовим для розробки ефективної та надійної системи.

Система повинна автоматично збирати дані про транзакції з блокчейн-мережі. Це включає отримання інформації про відправника, одержувача, суму, часову мітку та інші атрибути кожної транзакції. Дані повинні бути зібрані в режимі реального часу для забезпечення актуальності інформації.

На рисунку 2.1 зображена схема збору даних.

Зібрані дані повинні бути збережені у базі даних для подальшої обробки та аналізу. Система повинна забезпечувати збереження історичних даних про транзакції, а також їхню обробку, включаючи очищення, нормалізацію та перетворення у зручний для аналізу формат.

На рисунку 2.2 зображена схема обробки даних.

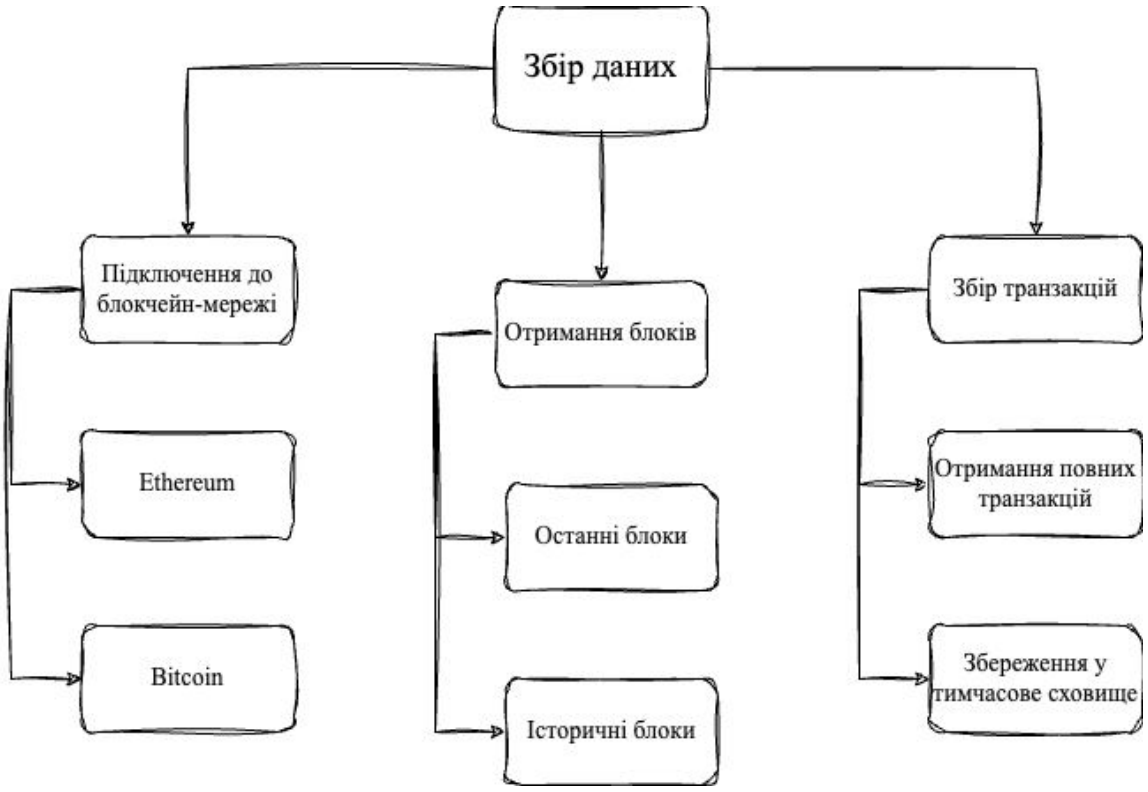

Рисунок 2.1 – Схема збору даних

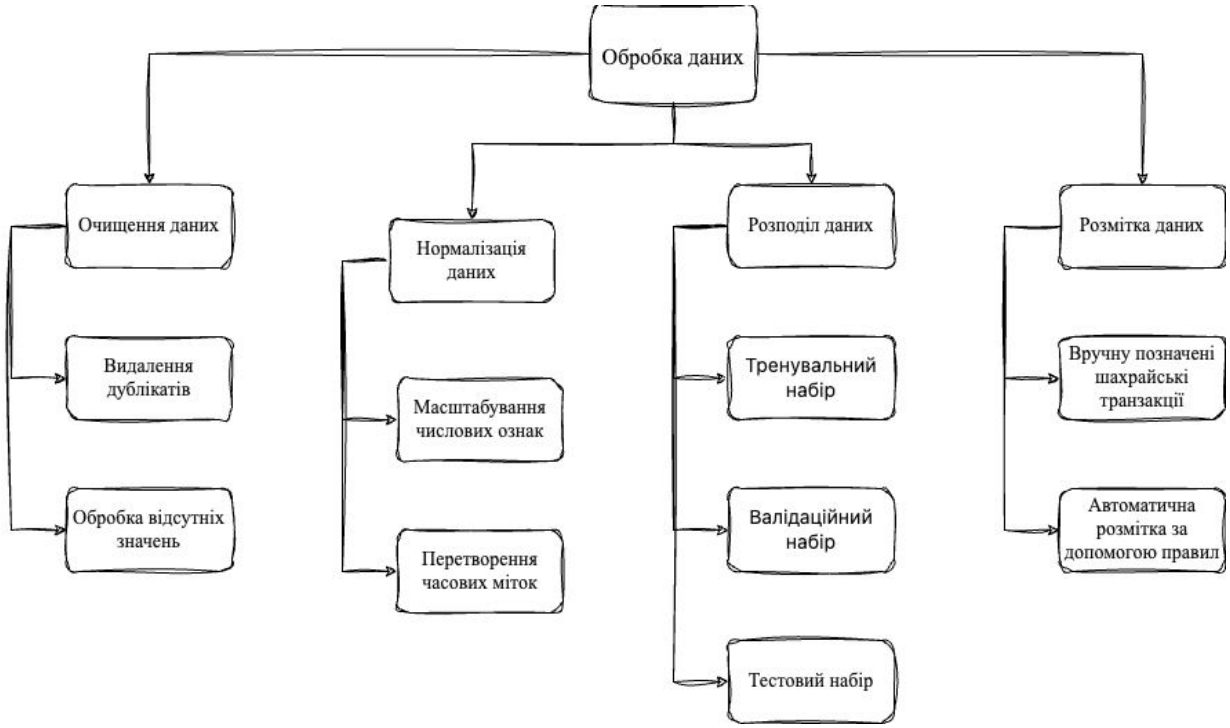

Рисунок 2.2 – Схема обробки даних

Система повинна виконувати аналіз зібраних даних для виявлення аномалій та підозрілих транзакцій. Це включає застосування методів машинного та глибокого навчання, таких як логістична регресія, дерева рішень, випадкові ліси, LSTM та CNN. Аналіз повинен виконуватися у реальному часі для забезпечення швидкого виявлення потенційних загроз.

Система повинна автоматично класифікувати транзакції як шахрайські або не шахрайські. Це включає використання моделей машинного навчання для визначення ймовірності шахрайства для кожної транзакції. Класифікація повинна бути точною та ефективною для зниження ризику помилкових спрацювань. На рисунку 2.3 зображена схема класифікації транзакцій.

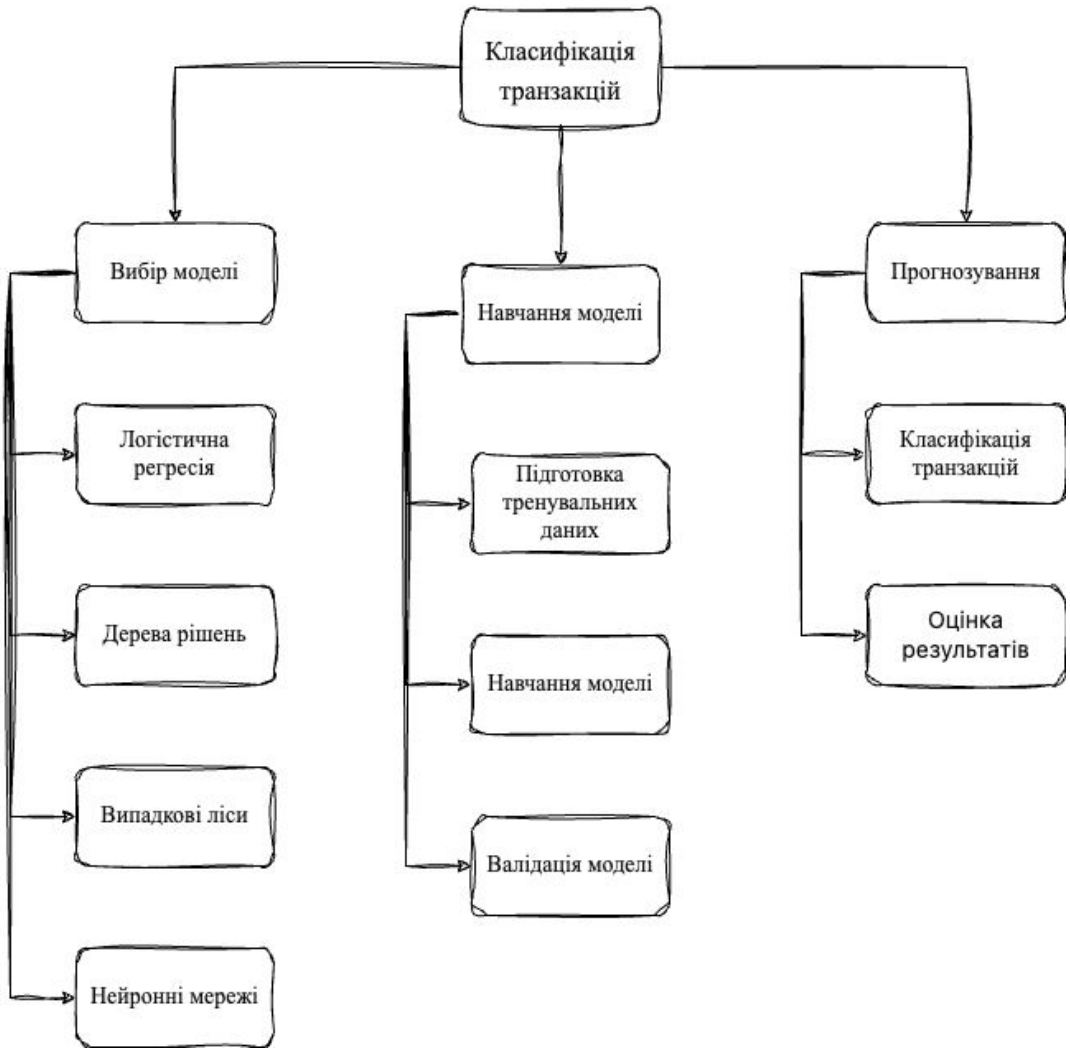

Рисунок 2.3 – Схема класифікації транзакцій

Система повинна забезпечувати моніторинг транзакцій у реальному часі. Це включає постійне відстеження нових транзакцій, їхній аналіз та класифікацію, а також оперативне реагування на виявлені аномалії. Моніторинг у реальному часі дозволяє швидко виявляти та запобігати шахрайству.

Система повинна надсилати нотифікації та алерти у разі виявлення підозрілих транзакцій. Нотифікації можуть бути надіслані відповідним користувачам або адміністраторам через різні канали, такі як електронна пошта, SMS або пуш-повідомлення. Це дозволяє оперативно реагувати на загрози та вживати необхідних заходів. На рисунку 2.4 зображена схема функціоналу нотифікацій.

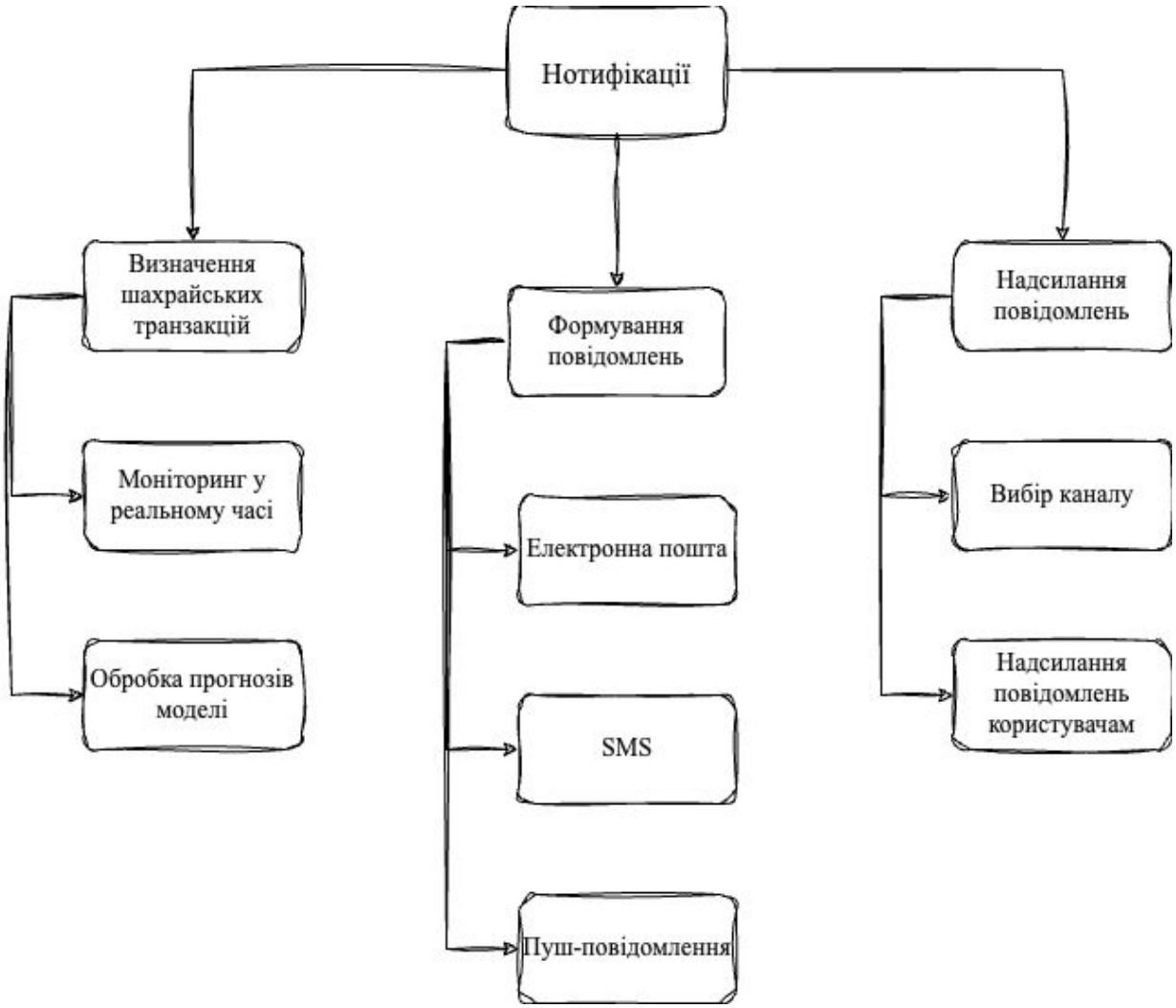

Рисунок 2.4 – Схема функціоналу нотифікацій

Система повинна автоматично створювати звіти про виявлені шахрайські транзакції. Звіти повинні включати детальну інформацію про транзакції, причини їх класифікації як шахрайських, а також рекомендації щодо подальших дій. Звіти можуть бути збережені у форматах, таких як PDF, Excel або інші, для зручності аналізу та зберігання. На рисунку 2.5 зображена схема функціоналу створення звітів.

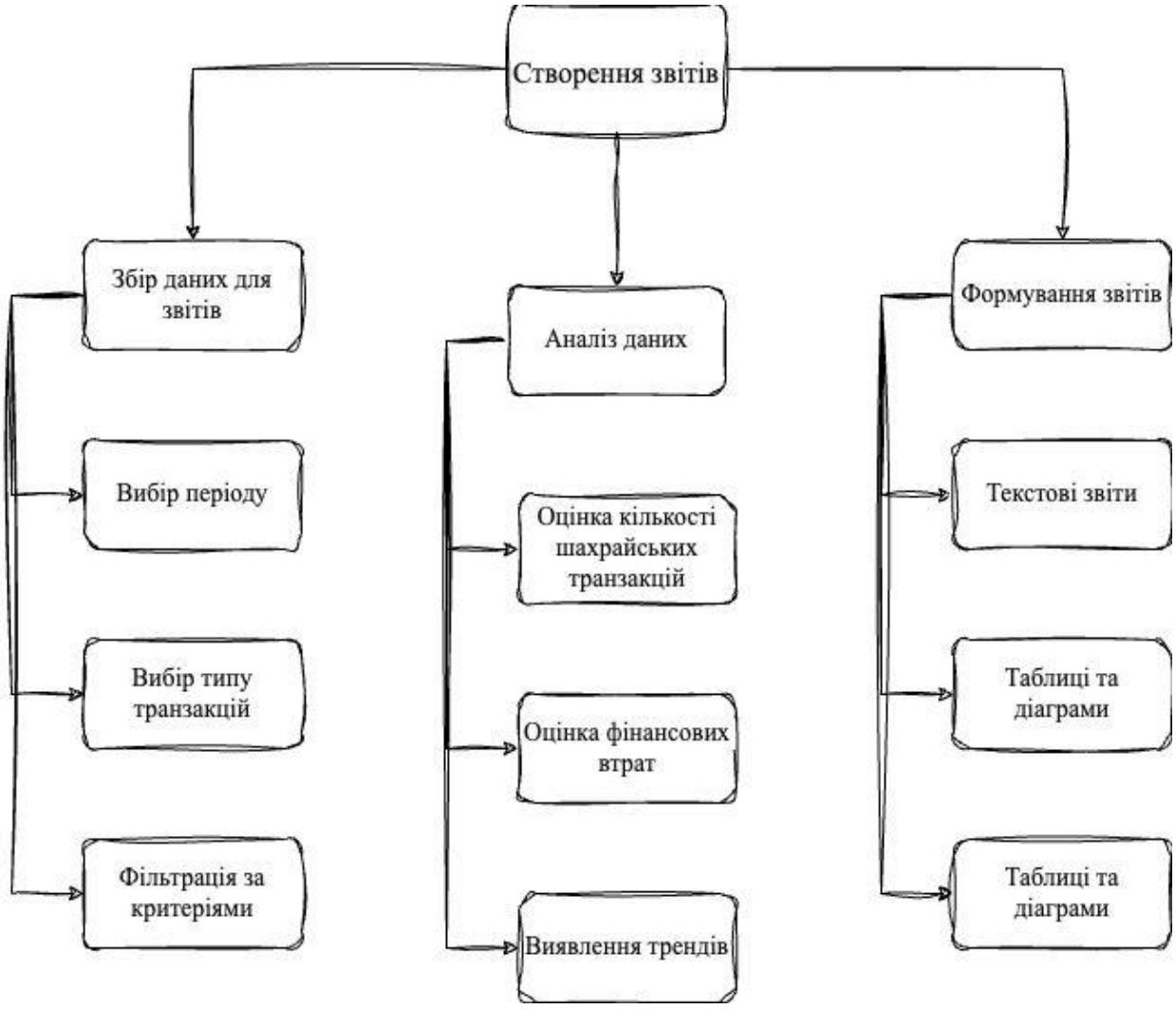

Рисунок 2.5 – Схема функціоналу створення звітів

Система повинна мати зручний інтерфейс користувача, який дозволяє здійснювати моніторинг та керування транзакціями. Інтерфейс повинен надавати користувачам можливість переглядати історію транзакцій,

аналізувати звіти, налаштовувати нотифікації та здійснювати інші операції. Важливо забезпечити інтуїтивно зрозумілий та простий у використанні інтерфейс.

На рисунку 2.6 зображена схема інтерфейсу користувача.

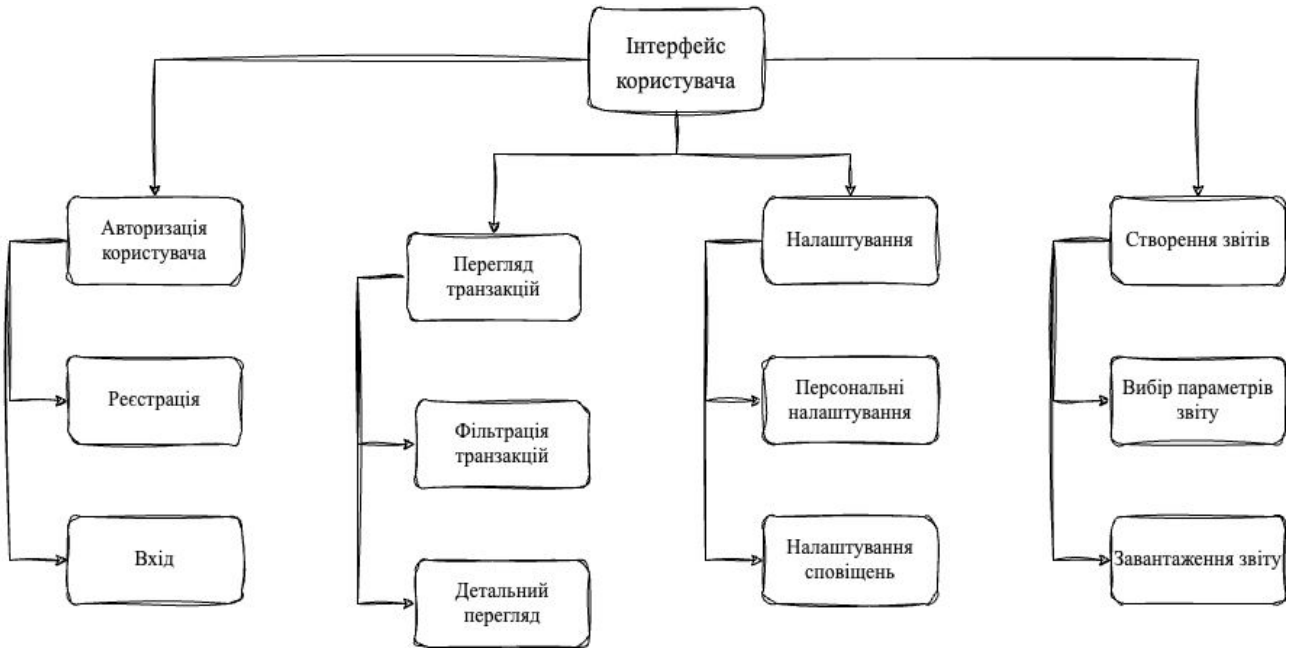

Рисунок 2.6 – Схема інтерфейсу користувача

Система повинна аналізувати поведінкові патерни користувачів для виявлення відхилень від нормальної поведінки. Це включає аналіз частоти та часу проведення транзакцій, типів адрес, з якими взаємодіють користувачі, та інших факторів. Відхилення від нормальної поведінки можуть свідчити про шахрайську активність.

Система повинна підтримувати інтеграцію з іншими системами, такими як платіжні шлюзи, криптовалютні гаманці та інші. Це дозволяє забезпечити комплексний підхід до моніторингу та виявлення шахрайських транзакцій. Інтеграція може здійснюватися через API або інші інтерфейси.

Система повинна бути масштабованою для обробки зростаючих обсягів даних та транзакцій. Це включає можливість додавання нових

серверів та ресурсів для забезпечення продуктивності та надійності системи при збільшенні навантаження.

Система повинна забезпечувати високий рівень безпеки для захисту даних та запобігання несанкціонованому доступу. Це включає використання методів шифрування, аутентифікації та авторизації, а також регулярні перевірки безпеки та оновлення програмного забезпечення.

На рисунку 2.7 зображена схема блоку «Безпека».

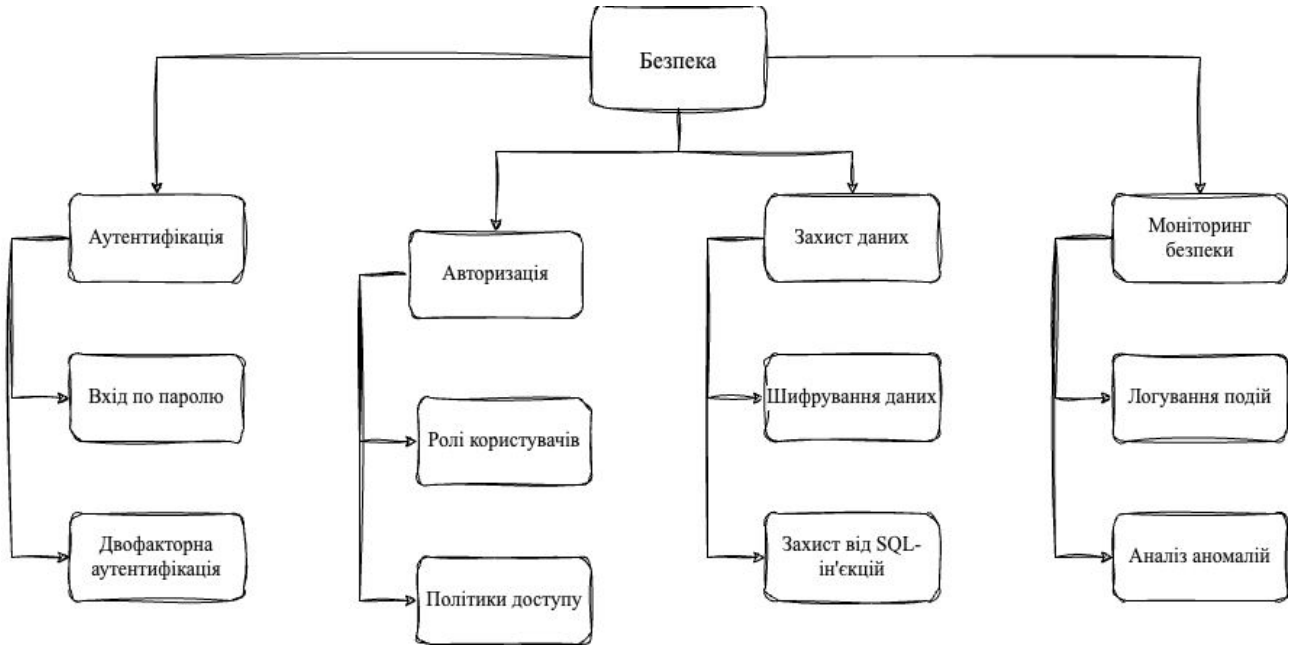

Рисунок 2.7 – Схема блоку «Безпека»

Система повинна підтримувати кілька мов для забезпечення зручності використання у різних регіонах та для різних користувачів. Це включає локалізацію інтерфейсу, повідомлень та звітів.

Система повинна мати механізми резервного копіювання та відновлення для запобігання втратам даних та забезпечення безперервності роботи. Це включає регулярне створення резервних копій баз даних та конфігурацій системи, а також тестування процедур відновлення.

Система повинна виконувати аналіз графів для виявлення підозрілих зв'язків між адресами та транзакціями. Це включає виявлення високого

ступеня зв'язності, циклічних структур та інших аномалій, що можуть свідчити про шахрайську активність.

На рисунку 2.8 зображена схема функціоналу аналіза графів.

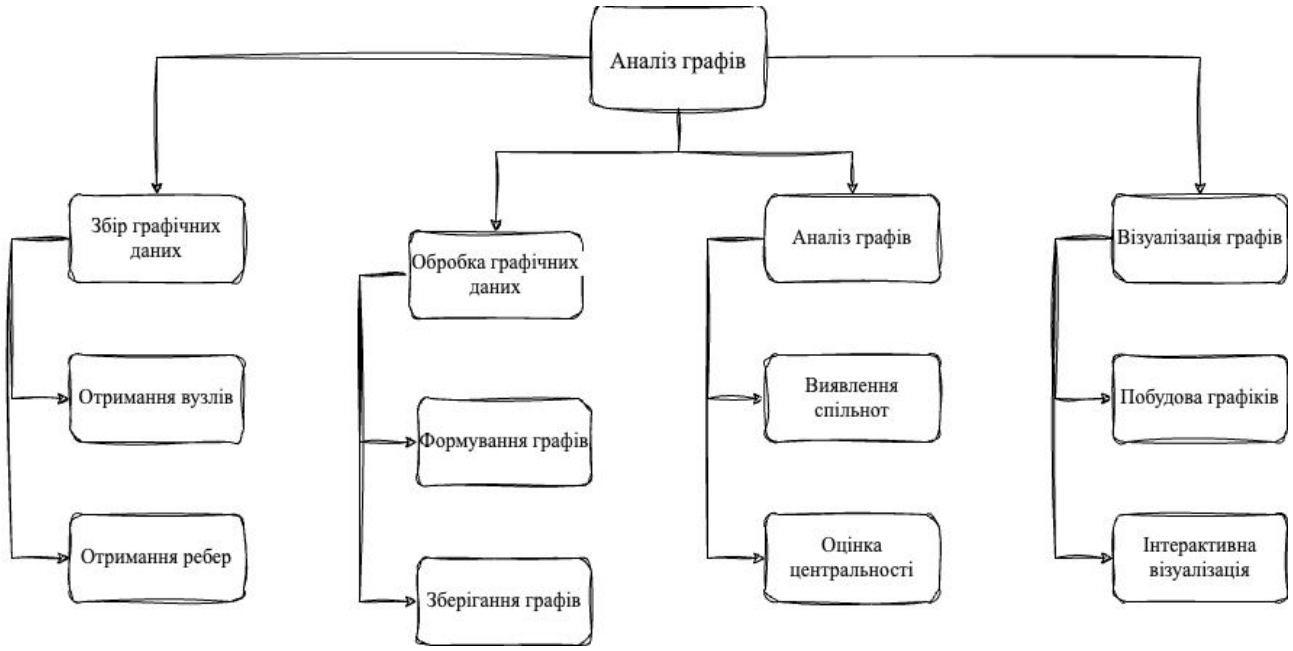

Рисунок 2.8 – Схема функціоналу аналіза графів

Система повинна підтримувати аналіз смарт-контрактів для виявлення потенційних вразливостей та шахрайських дій. Це включає автоматичний аудит смарт-контрактів та виявлення підозрілих транзакцій, пов'язаних зі смарт-контрактами.

Система повинна забезпечувати візуалізацію даних для зручності аналізу та прийняття рішень. Це включає графіки, діаграми та інші візуальні елементи, що дозволяють швидко оцінити стан транзакцій та виявити аномалії.

На рисунку 2.9 зображена схема функціоналу візуалізації даних.

Система повинна підтримувати роботу з різними блокчейн-платформами, такими як Ethereum, Bitcoin, Hyperledger та інші. Це забезпечує гнучкість та адаптивність системи для різних застосувань та вимог.

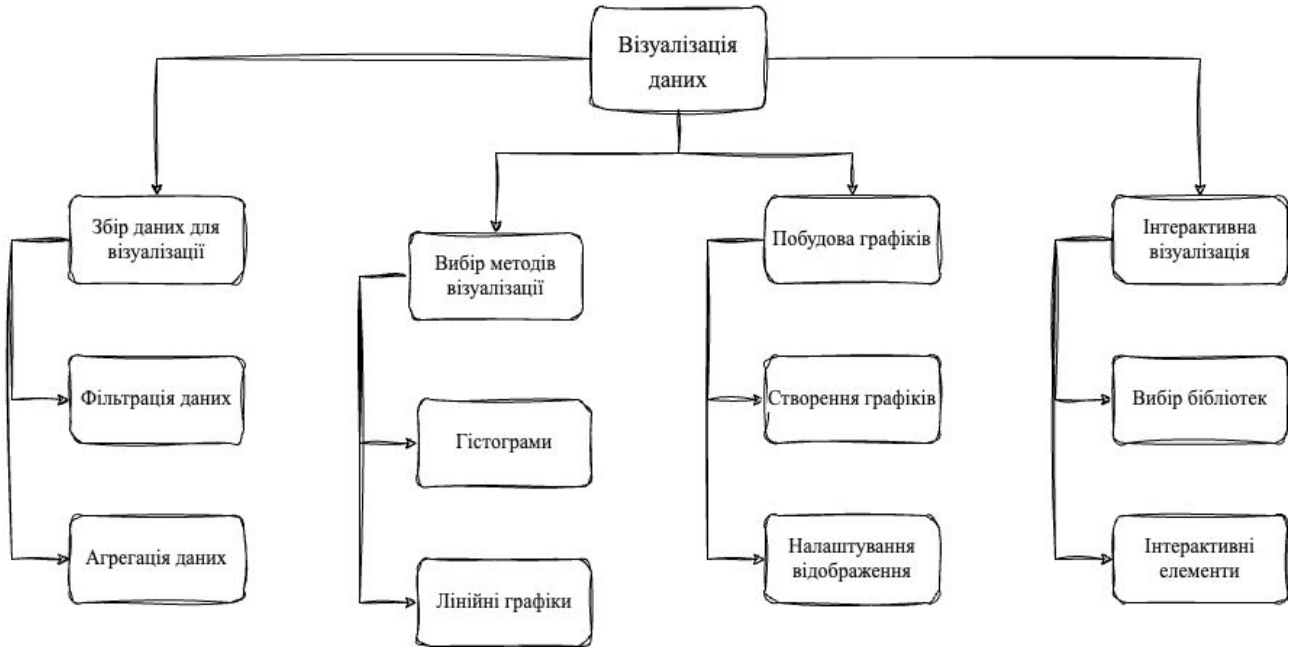

Рисунок 2.9 – Схема функціоналу візуалізації даних

Система повинна автоматично оновлювати моделі машинного та глибокого навчання для забезпечення їхньої актуальності та ефективності. Це включає регулярне перенавчання моделей на нових даних та адаптацію до змін у поведінці користувачів та транзакцій.

Система повинна підтримувати доступ з мобільних пристроїв для забезпечення зручності використання. Це включає мобільні додатки або адаптивний веб-інтерфейс, що дозволяють користувачам здійснювати моніторинг та керування транзакціями з мобільних пристроїв.

На рисунку 2.10 зображена схема мобільної підтримки.

Таким чином, функціональні вимоги до системи виявлення шахрайських транзакцій у блокчейні охоплюють широкий спектр можливостей та функцій. Вони забезпечують комплексний підхід до збору, обробки та аналізу даних, а також моніторингу та реагування на шахрайські транзакції. Чітке визначення та реалізація цих вимог дозволить створити ефективну та надійну систему, яка забезпечить захист фінансових операцій та зниження ризиків шахрайства.

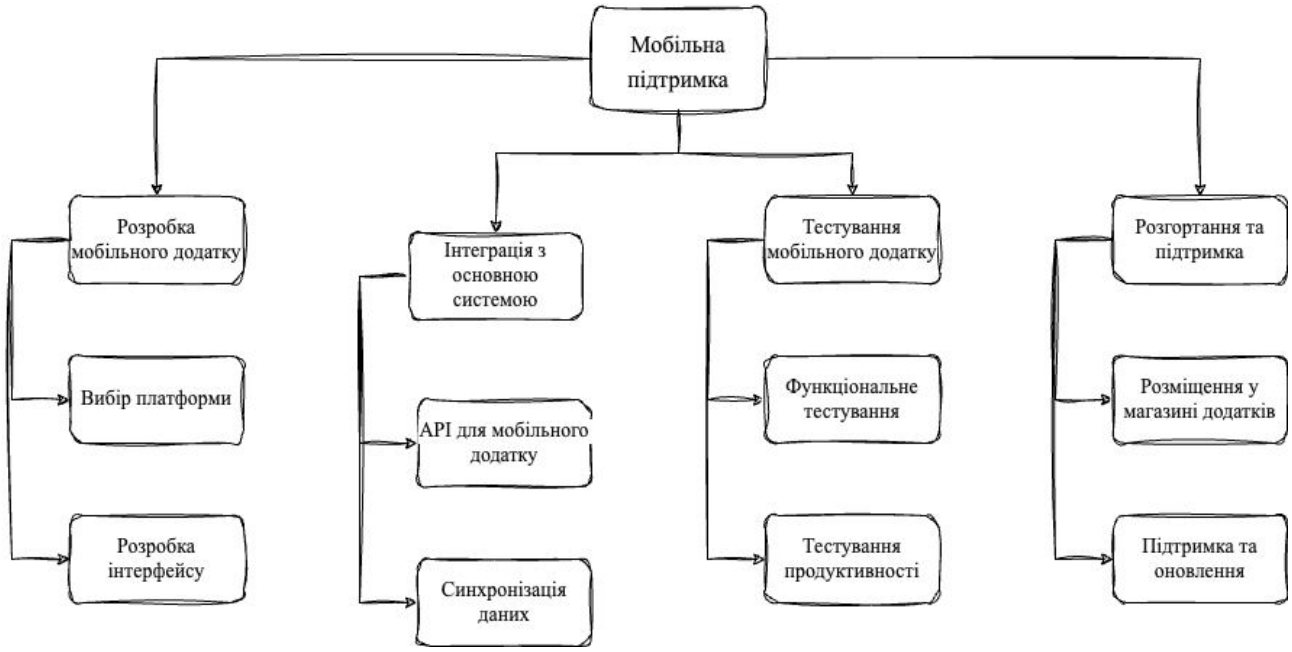

Рисунок 2.10 – Схема мобільної підтримки

2.3 Опис методів

Для виявлення шахрайських транзакцій у блокчейні застосовуються різні методи, включаючи машинне навчання, глибоке навчання, аналіз графів та поведінкові моделі. У цьому розділі розглянемо основні методи, що використовуються для виявлення аномалій та класифікації транзакцій як шахрайських або не шахрайських, з детальними поясненнями та формулами.

Логістична регресія.

Логістична регресія є одним з основних методів для бінарної класифікації, який широко використовується для виявлення шахрайських транзакцій. Логістична регресія моделює ймовірність події, зокрема шахрайства, використовуючи лінійну комбінацію ознак транзакції.

Формула для логістичної регресії:

$$\mathrm{P}(Y = 1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_n X_n)}}, \quad (2.1)$$

де $P(Y = 1|X)$ – ймовірність того, що транзакція є шахрайською;

β_0 – вільний член;

$\beta_1 + \beta_2 + \cdots + \beta_n$ – коефіцієнти регресії;

$X_1 + X_2 + \cdots + X_n$ – ознаки транзакції.

Дерева рішень.

Дерева рішень використовують структуру дерева для прийняття рішень на основі ознак транзакцій. Кожен вузол дерева представляє ознаку, кожна гілка – умову на цю ознаку, а кожен лист – клас (шахрайська або не шахрайська транзакція).

Приклад дерева рішень для виявлення шахрайських транзакцій представлений на рисунку 2.11.

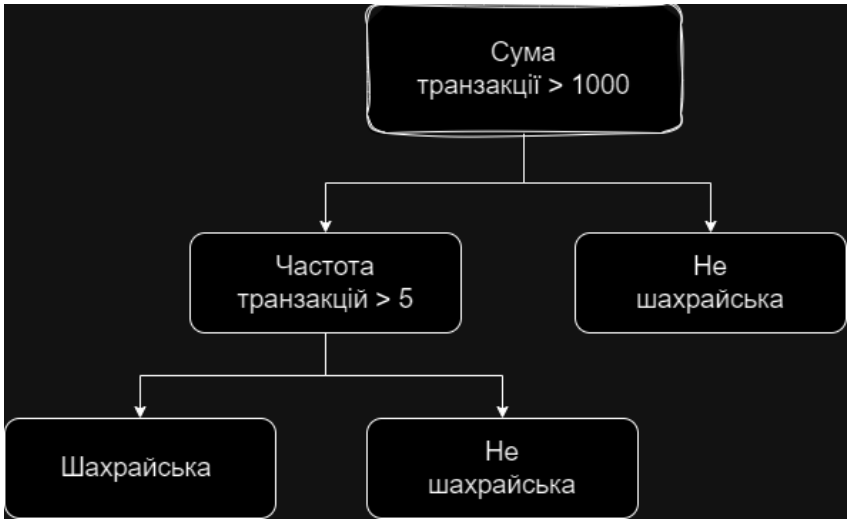

Рисунок 2.11 – Приклад дерева рішень для виявлення шахрайських транзакцій

Алгоритм побудови дерева рішень включає вибір найбільш значущих ознак та створення умов, що мінімізують кількість помилок класифікації.

Випадкові ліси.

Випадкові ліси складаються з множини дерев рішень, кожне з яких будується на випадковій підмножині даних та ознак. Кінцеве рішення приймається шляхом голосування всіх дерев.

Формула для обчислення прогнозу у випадковому лісі:

$$\check{y} = \frac{1}{M}\sum_{m=1}^{M} h_m(x), \tag{2.2}$$

де M – кількість дерев у лісі;

$h_m(x)$ – прогноз m-го дерева для прикладу x.

Рекурентні нейронні мережі (RNN).

Рекурентні нейронні мережі є потужними інструментами для аналізу послідовних даних, таких як транзакції у блокчейні. LSTM (Long Short-Term Memory) є спеціальним типом RNN, який здатен зберігати та оновлювати інформацію протягом довгих періодів часу.

Формули для LSTM:

– ворота забування:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f; \tag{2.3}$$

– ворота введення:

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i; \tag{2.4}$$

– оновлення стану пам'яті:

$$\widetilde{C_t} = \tanh(W_C * [h_{t-1}, x_t] + b_C; \tag{2.5}$$

– новий стан пам'яті:

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}; \tag{2.6}$$

– ворота виходу:

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o); \quad (2.7)$$

– новий вихід:

$$h_t = o_t * \tanh(C_t). \quad (2.8)$$

Конволюційні нейронні мережі (CNN).

CNN здатні автоматично вилучати складні ознаки з даних і можуть бути ефективними для аналізу структурованих даних, таких як графи транзакцій. Основними компонентами CNN є конволюційні шари, шари підсумування (pooling) та повнозв'язні шари.

Формула для конволюційного шару:

$$h_{i,j}^k = \tanh(\sum_{m,n} x_{i+m,j+n} * w_{m,n}^k + b^k), \quad (2.9)$$

де $h_{i,j}^k$ – вихід нейрона на позиції i, j у k -му каналі;

$x_{i+m,j+n}$ – значення вхідного нейрона на позиції $i + m, j + n$;

$w_{m,n}^k$ – вага фільтра;

b^k – зміщення.

Аналіз графів.

Блокчейн може бути представлений у вигляді графу, де вузли представляють адреси або користувачів, а ребра – транзакції між ними. Аналіз графів дозволяє виявляти підозрілі зв'язки між вузлами, такі як високий ступінь зв'язності або циклічні структури.

Алгоритм виявлення спільнот Louvain:

– початкова модульність:

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j); \quad (2.10)$$

– оновлення спільнот:

$$\acute{Q} = Q + \sum_i [\frac{e_i + k_{i,in}}{2m} - \left(\frac{e_i + k_i}{2m}\right)^2]. \quad (2.11)$$

де Q – модульність;

A_{ij} – елемент матриці суміжності;

k_i та k_j – ступені вузлів і та j;

m – кількість ребер у графі;

$\delta(c_i, c_j)$ – функція Кронекер;

e_i – внутрішні ребра у спільноті і;

$k_{i,in}$ – внутрішній ступінь вузла і.

Кластеризація (K-середніх):

K-середніх (K-means) – це метод кластеризації, який розбиває дані на k кластерів, мінімізуючи відстань між точками всередині кожного кластера та їхнім центроїдом.

Формула для оновлення центроїдів у алгоритмі K-means:

$$\mu_i = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j, \quad (2.12)$$

де μ_i – центроїд і-го кластера;

C_i – набір точок у і-му кластері;

x_j – точка даних.

Аномалійне виявлення (Isolation Forest).

Метод відокремлення (Isolation Forest) базується на ідеї, що аномальні точки знаходяться на меншій глибині у дереві, що відокремлює дані.

Формула для обчислення аномалійного оцінювання у методі відокремлення:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}, \quad (2.13)$$

де $s(x,n)$ – аномалійне оцінювання точки x у наборі даних з n точок;

$E(h(x))$ – середня глибина точки x у дереві відокремлення;

c(n) – середня глибина випадкового дерева відокремлення.

Метод підтримкових векторів (One-Class SVM).

One-Class SVM намагається знайти гіперплощину, що відокремлює більшість даних від аномалій.

Формула для One-Class SVM:

$$min_{w,p}\frac{1}{2}\|w\|^2 + \frac{1}{vn}\sum_{i=1}^{n}\max(0, p - w * x_i) - p, \quad (2.14)$$

де w – вектор ваг;

p – відступ (offset);

v – параметр регуляризації;

x_i – точка даних.

Баєсівські мережі.

Баєсівські мережі є графічними моделями, які використовуються для представлення залежностей між змінними. Вони можуть бути використані для виявлення шахрайських транзакцій, враховуючи ймовірнісні залежності між різними атрибутами транзакцій.

Формула для обчислення апостеріорної ймовірності у Баєсівській мережі:

$$P\big(X_i|\text{батьки}(X_i)\big) = \frac{P(\text{батьки}(X_i)|X_i)P(X_i)}{P(\text{батьки}(X_i))}, \quad (2.15)$$

де X_i – змінна;

батьки(X_i) – набір батьків змінної X_i.

Графічні нейронні мережі (GNN).

GNN використовуються для аналізу графів і можуть бути ефективними для виявлення шахрайства у блокчейн-транзакціях. GNN можуть автоматично вилучати ознаки з графів та здійснювати класифікацію вузлів або ребер.

Формула для обчислення оновленого стану вузла у GNN:

$$h_v^{(k+1)} = \sigma(W * \sum_{u \epsilon N(v)} \frac{1}{C_{vu}} h_u^{(k)} + b), \quad (2.16)$$

де $h_v^{(k+1)}$ – оновлений стан вузла v на $(k+1)$-му шарі;

$N(v)$ – сусіди вузла v;

C_{vu} – нормувальний коефіцієнт;

W – матриця ваг;

b – вектор зміщення;

σ – активаційна функція.

Таким чином, існує багато методів, які можуть бути використані для виявлення шахрайських транзакцій у блокчейні. Кожен метод має свої переваги та недоліки, і вибір методу залежить від специфіки завдання, типу даних та вимог до точності та продуктивності. Комбінація різних методів може забезпечити більш точне та ефективне виявлення шахрайства.

2.4 Уточнення методів

Уточнення методів для виявлення шахрайських транзакцій у блокчейні включає детальний опис кожного методу, його параметрів, гіперпараметрів, процедур навчання, тестування та оцінки ефективності. У цьому розділі ми розглянемо, як кожен метод може бути налаштований і оптимізований для досягнення максимальних результатів.

Логістична регресія є одним із найбільш простих і ефективних методів для бінарної класифікації. Для налаштування логістичної регресії необхідно вибрати відповідні ознаки та налаштувати гіперпараметри:

– ознаки (features): вибір ознак є критично важливим для успішної роботи моделі. Ознаки можуть включати суму транзакції, частоту транзакцій, час проведення транзакцій, адреси відправника та одержувача та інші релевантні фактори;

– гіперпараметри: основним гіперпараметром логістичної регресії є параметр регуляризації C, який контролює баланс між точністю моделі та її здатністю до узагальнення;

– навчання: для навчання моделі використовується метод максимального правдоподібності, який оптимізує логістичну функцію втрат;

– оцінка: ефективність моделі оцінюється за допомогою метрик, таких як точність (accuracy), повнота (recall), специфічність (specificity) та F1-міра.

Дерева рішень є потужним методом для класифікації та регресії. Вони побудовані шляхом рекурсивного поділу даних на підмножини на основі значень ознак:

– ознаки: ознаки можуть включати суму транзакції, час проведення, частоту, адреси та інші релевантні фактори;

– гіперпараметри: основні гіперпараметри дерева рішень включають глибину дерева (max depth), мінімальну кількість зразків для поділу вузла (min samples split) та мінімальну кількість зразків у листі (min samples leaf);

– навчання: дерево будується шляхом вибору найкращих розділів на основі критерію, такого як Gini або ентропія;

– оцінка: ефективність оцінюється за допомогою крос-валідації та метрик, таких як точність, повнота та специфічність.

Випадкові ліси є ансамблевим методом, що складається з множини дерев рішень. Вони забезпечують високу точність і стійкість до перенавчання:

– ознаки: використовуються ті ж ознаки, що і для дерев рішень;

– гіперпараметри: основні гіперпараметри включають кількість дерев у лісі (n estimators), максимальну глибину дерев, мінімальну кількість зразків для поділу вузла та мінімальну кількість зразків у листі;

– навчання: кожне дерево будується на випадковій підмножині даних та ознак;

– оцінка: оцінка проводиться за допомогою крос-валідації та метрик, таких як точність, повнота та специфічність.

RNN та LSTM використовуються для аналізу послідовних даних, таких як транзакції, що відбуваються з часом:

– ознаки: послідовності транзакційних даних, що включають суми, час проведення, адреси та інші фактори;

– гіперпараметри: основні гіперпараметри включають кількість шарів LSTM, кількість нейронів у кожному шарі, швидкість навчання, розмір міні-пакету (batch size) та кількість епох;

– навчання: використовується метод зворотного поширення помилки через час (BPTT) для навчання мережі;

– оцінка: модель оцінюється за допомогою метрик, таких як точність, повнота та специфічність, на тестових даних.

CNN використовуються для вилучення ознак з графів транзакцій та класифікації їх як шахрайських або не шахрайських:

– ознаки: графічні представлення транзакційних даних;

– гіперпараметри: основні гіперпараметри включають кількість конволюційних шарів, кількість фільтрів у кожному шарі, розмір фільтрів, розмір підсумкових шарів та кількість повнозв'язних шарів;

– навчання: використовується метод зворотного поширення помилки для навчання мережі;

– оцінка: модель оцінюється за допомогою метрик, таких як точність, повнота та специфічність, на тестових даних.

Аналіз графів дозволяє виявляти підозрілі зв'язки між адресами та транзакціями:

– ознаки: вузли та ребра графу, що представляють адреси та транзакції;

– гіперпараметри: основні гіперпараметри включають кількість ітерацій алгоритму, тип алгоритму (наприклад, PageRank, Louvain) та параметри нормалізації;

– навчання: використовується алгоритм виявлення спільнот або центральності для аналізу графу;

– оцінка: оцінка проводиться за допомогою метрик, таких як модульність (для алгоритму Louvain) або значення центральності (для алгоритму PageRank).

Кластеризація використовується для групування транзакцій та виявлення аномалій:

– ознаки: транзакційні дані, такі як сума, частота, час проведення;

– гіперпараметри: основний гіперпараметр – кількість кластерів k;

– навчання: використовується алгоритм K-середніх для розподілу даних на кластери;

– оцінка: оцінка проводиться за допомогою метрик, таких як сумарна внутрішньокластерна відстань або коефіцієнт Сілуета.

Метод Isolation Forest використовується для виявлення аномальних транзакцій:

– ознаки: транзакційні дані, такі як сума, частота, час проведення;

– гіперпараметри: основні гіперпараметри включають кількість дерев у лісі (n estimators) та максимальну глибину дерев;

– навчання: кожне дерево будується на випадковій підмножині даних, а аномалії виявляються на основі глибини відокремлення;

– оцінка: оцінка проводиться за допомогою метрик, таких як точність, повнота та специфічність.

One-Class SVM використовується для виявлення аномалій:

– ознаки: транзакційні дані, такі як сума, частота, час проведення;

– гіперпараметри: основні гіперпараметри включають параметр регуляризації v та тип ядра (kernel);

– навчання: використовується метод градієнтного спуску для оптимізації моделі;

– оцінка: оцінка проводиться за допомогою метрик, таких як точність, повнота та специфічність.

Баєсівські мережі використовуються для моделювання ймовірнісних залежностей між ознаками:

– ознаки: транзакційні дані, такі як сума, частота, час проведення;

– гіперпараметри: основні гіперпараметри включають структуру мережі та ймовірнісні параметри;

– навчання: використовується метод максимального правдоподібності або байєсівська оцінка для навчання мережі;

– оцінка: оцінка проводиться за допомогою метрик, таких як точність, повнота та специфічність.

GNN використовуються для аналізу графів транзакцій та виявлення шахрайства:

– ознаки: вузли та ребра графу, що представляють адреси та транзакції;

– гіперпараметри: основні гіперпараметри включають кількість шарів, кількість нейронів у кожному шарі, швидкість навчання та кількість епох;

– навчання: використовується метод зворотного поширення помилки для навчання мережі;

– оцінка: оцінка проводиться за допомогою метрик, таких як точність, повнота та специфічність.

Таким чином, уточнення методів включає детальний опис кожного методу, його параметрів, процедур навчання та оцінки. Вибір відповідних методів та їх налаштування є критично важливими для досягнення високої точності та ефективності системи виявлення шахрайських транзакцій у блокчейні. Кожен метод має свої переваги та недоліки, тому комбінування кількох методів може забезпечити кращі результати.

2.5 Опис технології

Опис технології включає детальний розгляд усіх компонентів системи, її архітектуру, використовувані інструменти та фреймворки, а також методи інтеграції та взаємодії між компонентами. У цьому розділі буде надано комплексний огляд технології, яка використовується для виявлення шахрайських транзакцій у блокчейні, включаючи апаратне забезпечення, програмне забезпечення, алгоритми, бази даних, та інші інструменти.

Архітектура системи.

Архітектура системи включає такі основні компоненти:

– збір даних: компонент для автоматичного збирання даних про транзакції з блокчейн-мережі;

– збереження даних: база даних для зберігання транзакційних даних та історії транзакцій;

– аналіз даних: модулі для аналізу та класифікації транзакцій з використанням методів машинного та глибокого навчання;

– моніторинг: інтерфейс для моніторингу транзакцій у реальному часі та виявлення аномалій;

– нотифікації: система для надсилання повідомлень про виявлені шахрайські транзакції;

– звітування: інструменти для автоматичного створення звітів.

Архітектура системи може бути представлена у вигляді схеми (рисунок 2.12):

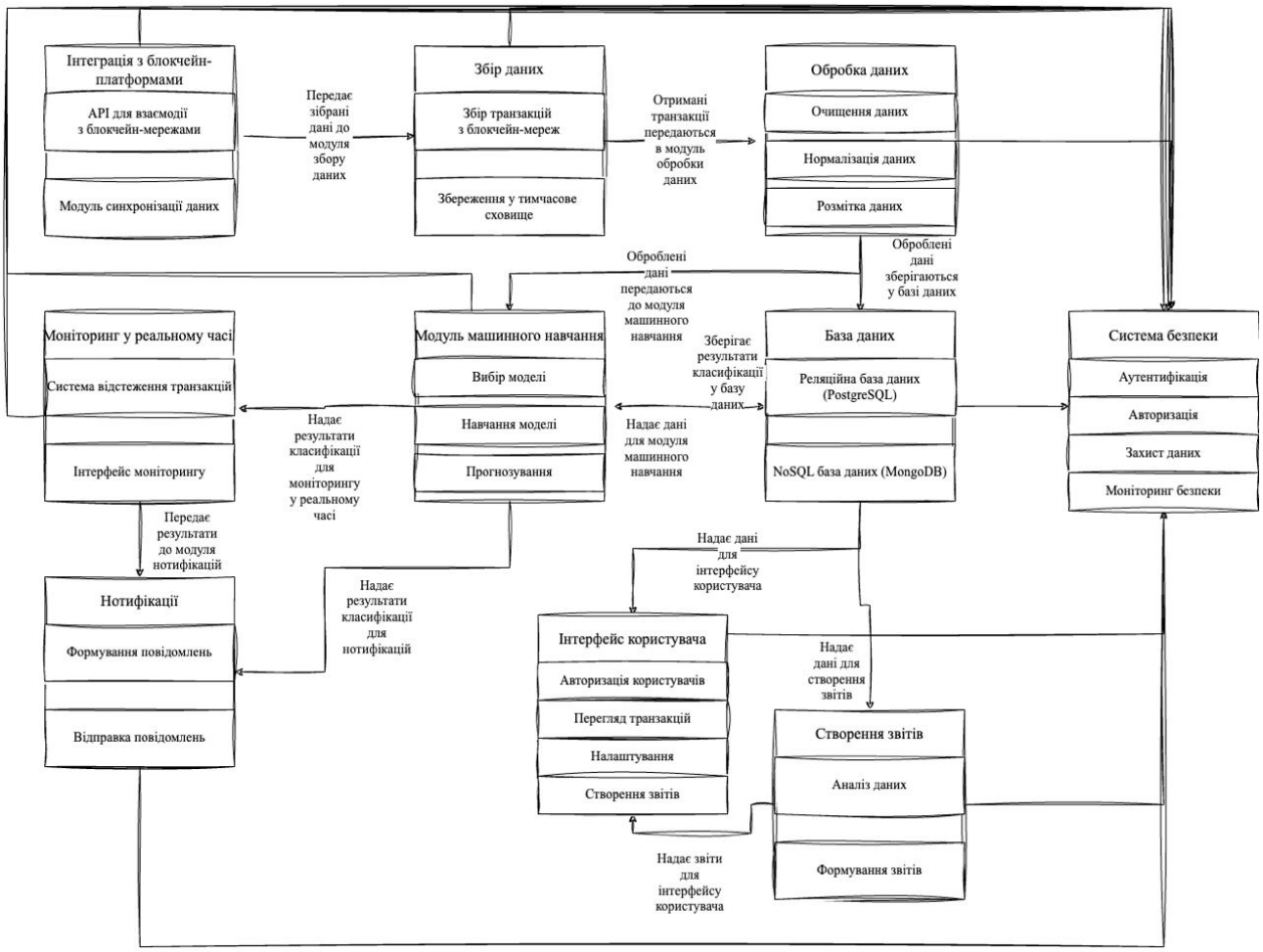

Рисунок 2.12 – Схема архітектури системи

Збір даних.

Компонент збору даних відповідає за отримання інформації про транзакції з блокчейн-мережі. Це може бути реалізовано за допомогою API конкретної блокчейн-платформи (наприклад, Ethereum або Bitcoin). Компонент повинен забезпечувати збір даних у режимі реального часу, а також мати можливість оновлення історичних даних.

Інструменти: Web3.js для взаємодії з Ethereum, Bitcoin RPC API для взаємодії з Bitcoin.

Збереження даних.

Для зберігання транзакційних даних використовується база даних. Рекомендується використовувати реляційні бази даних, такі як PostgreSQL або MySQL, для зберігання структурованих даних. Для обробки великих обсягів неструктурованих даних та швидкого пошуку можна використовувати NoSQL бази даних, такі як MongoDB або Cassandra.

Інструменти: PostgreSQL, MySQL, MongoDB, Cassandra.

Аналіз даних.

Аналіз даних включає використання алгоритмів машинного та глибокого навчання для виявлення шахрайських транзакцій. Модулі аналізу повинні бути здатні обробляти великі обсяги даних та виконувати аналіз у режимі реального часу.

Інструменти: TensorFlow, PyTorch, Scikit-learn, Dask.

Моніторинг у реальному часі.

Моніторинг у реальному часі дозволяє постійно відстежувати нові транзакції, їхній аналіз та класифікацію. Інтерфейс моніторингу повинен бути зручним та інтуїтивно зрозумілим, забезпечуючи швидкий доступ до необхідної інформації та можливість реагування на виявлені аномалії.

Інструменти: React.js для створення користувацького інтерфейсу, Node.js для серверної частини, WebSockets для реального часу.

Нотифікації.

Система нотифікацій надсилає повідомлення про виявлені шахрайські транзакції відповідним користувачам або адміністраторам. Нотифікації можуть бути надіслані через різні канали, такі як електронна пошта, SMS або пуш-повідомлення.

Інструменти: Twilio для SMS-повідомлень, SendGrid для електронної пошти, Firebase для пуш-повідомлень.

Звітування.

Система повинна автоматично створювати звіти про виявлені шахрайські транзакції. Звіти повинні містити детальну інформацію про транзакції, причини їх класифікації як шахрайських та рекомендації щодо

подальших дій. Звіти можуть бути збережені у форматах, таких як PDF, Excel або інші.

Інструменти: Apache POI для створення Excel-звітів, iText для створення PDF-звітів.

Апаратне забезпечення.

Для забезпечення ефективної роботи системи необхідно використовувати потужні апаратні ресурси. Це включає потужні процесори (CPU), графічні процесори (GPU) для прискорення обчислень, великі обсяги оперативної пам'яті (RAM) та швидкі накопичувачі даних (SSD).

Рекомендації: Intel Xeon або AMD EPYC для CPU, NVIDIA Tesla або AMD Instinct для GPU, мінімум 16 GB RAM, мінімум 1 TB SSD.

Фреймворки для машинного та глибокого навчання.

Для розробки моделей машинного та глибокого навчання використовуються сучасні фреймворки, які забезпечують широкі можливості для створення, навчання та розгортання нейронних мереж.

Фреймворки: TensorFlow, PyTorch, Keras, Scikit-learn.

Інструменти для оркестрації та розгортання.

Для забезпечення масштабованості та керованості системи використовуються інструменти для оркестрації та розгортання контейнерів. Це дозволяє автоматично масштабувати ресурси та керувати розгортанням сервісів.

Інструменти: Docker для контейнеризації, Kubernetes для оркестрації контейнерів.

Інструменти для моніторингу та логування.

Для забезпечення стабільної роботи системи необхідно впровадити інструменти для моніторингу та логування. Це включає збір та аналіз метрик продуктивності, логування подій та виявлення помилок.

Інструменти: Prometheus для збору та аналізу метрик, Grafana для візуалізації даних, ELK Stack (Elasticsearch, Logstash, Kibana) для логування.

Забезпечення безпеки.

Безпека є критично важливим аспектом для системи виявлення шахрайських транзакцій. Необхідно забезпечити захист даних та системи від несанкціонованого доступу, використовуючи методи шифрування, аутентифікації та авторизації.

Інструменти: TLS для шифрування даних, OAuth для аутентифікації та авторизації, регулярні перевірки безпеки та оновлення програмного забезпечення.

Резервне копіювання та відновлення.

Для запобігання втратам даних та забезпечення безперервності роботи системи необхідно впровадити механізми резервного копіювання та відновлення. Це включає регулярне створення резервних копій баз даних та конфігурацій системи, а також тестування процедур відновлення.

Інструменти: Bacula для резервного копіювання, сценарії для автоматизації процесів резервного копіювання та відновлення.

Інтеграція з іншими системами.

Система повинна підтримувати інтеграцію з іншими системами, такими як платіжні шлюзи, криптовалютні гаманці та інші. Це дозволяє забезпечити комплексний підхід до моніторингу та виявлення шахрайських транзакцій.

Інструменти: RESTful API для інтеграції, веб-хуки для взаємодії з іншими системами.

Підтримка мобільних пристроїв.

Система повинна підтримувати доступ з мобільних пристроїв для забезпечення зручності використання. Це включає мобільні додатки або адаптивний веб-інтерфейс, що дозволяють користувачам здійснювати моніторинг та керування транзакціями з мобільних пристроїв.

Інструменти: React Native для розробки мобільних додатків, адаптивний дизайн для веб-інтерфейсу.

Візуалізація даних.

Система повинна забезпечувати візуалізацію даних для зручності аналізу та прийняття рішень. Це включає графіки, діаграми та інші візуальні елементи, що дозволяють швидко оцінити стан транзакцій та виявити аномалії.

Інструменти: D3.js для візуалізації даних, Chart.js для створення графіків та діаграм.

Підтримка кількох мов.

Система повинна підтримувати кілька мов для забезпечення зручності використання у різних регіонах та для різних користувачів. Це включає локалізацію інтерфейсу, повідомлень та звітів.

Інструменти: i18next для локалізації інтерфейсу, gettext для перекладу текстових повідомлень.

Автоматичне оновлення моделей.

Система повинна автоматично оновлювати моделі машинного та глибокого навчання для забезпечення їхньої актуальності та ефективності. Це включає регулярне перенавчання моделей на нових даних та адаптацію до змін у поведінці користувачів та транзакцій.

Інструменти: Airflow для автоматизації робочих процесів з перенавчання моделей, Jenkins для безперервної інтеграції та доставки.

Забезпечення надійності.

Для забезпечення надійності системи необхідно використовувати відмовостійкі архітектурні підходи, такі як реплікація баз даних, балансування навантаження та автоматичне масштабування.

Інструменти: HAProxy для балансування навантаження, Kubernetes для автоматичного масштабування, PostgreSQL Streaming Replication для реплікації баз даних.

Підтримка різних блокчейн-платформ.

Система повинна підтримувати роботу з різними блокчейн-платформами, такими як Ethereum, Bitcoin, Hyperledger та інші. Це

забезпечує гнучкість та адаптивність системи для різних застосувань та вимог.

Інструменти: Web3.js для взаємодії з Ethereum, Bitcoin RPC API для взаємодії з Bitcoin, Hyperledger Fabric SDK для інтеграції з Hyperledger.

Таким чином, опис технології включає детальний розгляд усіх компонентів системи, їхню архітектуру, використовувані інструменти та фреймворки. Врахування всіх аспектів, від апаратного забезпечення до методів інтеграції, допоможе забезпечити ефективну та надійну роботу системи для виявлення шахрайських транзакцій у блокчейні.

3 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

3.1 Вибір технологій

Вибір технологій є одним з найважливіших етапів розробки системи для виявлення шахрайських транзакцій у блокчейні. Правильний вибір технологій визначає ефективність, надійність та масштабованість системи. У цьому розділі буде детально розглянуто вибір апаратного та програмного забезпечення, мови програмування, фреймворків, баз даних та інших інструментів, які будуть використовуватися у проекті.

Мова програмування.

Для реалізації системи необхідно вибрати мову програмування, яка забезпечить високу продуктивність та підтримку сучасних бібліотек для машинного навчання та аналізу даних. Найпопулярнішими мовами для таких завдань є Python, R, Java та C++.

Python: Python є найпопулярнішою мовою для розробки систем машинного навчання завдяки своїй простоті та багатству бібліотек, таких як TensorFlow, PyTorch, Scikit-learn, Pandas та NumPy. Python забезпечує швидку розробку та легку інтеграцію з іншими системами.

R: R є потужною мовою для статистичного аналізу та візуалізації даних. Вона має широкий набір пакетів для машинного навчання та аналізу даних, таких як caret, randomForest, e1071 та інші.

Java: Java є потужною мовою для розробки корпоративних додатків та систем, що потребують високої продуктивності та надійності. Java має бібліотеки для машинного навчання, такі як Weka, Deeplearning4j та MOA.

C++: C++ забезпечує високу продуктивність та контроль над апаратними ресурсами. Він використовується для розробки високопродуктивних додатків та систем, що потребують мінімальних затримок. C++ має бібліотеки для машинного навчання, такі як Dlib, Shark та mlpack.

Фреймворки для машинного та глибокого навчання.

Для розробки моделей машинного та глибокого навчання використовуються сучасні фреймворки, які забезпечують широкі можливості для створення, навчання та розгортання нейронних мереж.

TensorFlow: TensorFlow є одним з найпопулярніших фреймворків для глибокого навчання, розроблених Google. Він підтримує побудову складних нейронних мереж, навчання на великих наборах даних та розгортання на різних платформах.

PyTorch: PyTorch є іншим популярним фреймворком для глибокого навчання, розробленим Facebook. Він забезпечує гнучкість та зручність у використанні завдяки динамічному обчислювальному графу та простому інтерфейсу.

Keras: Keras є високорівневим API для створення нейронних мереж, який працює поверх TensorFlow або Theano. Keras забезпечує швидку розробку моделей завдяки простому та інтуїтивно зрозумілому інтерфейсу.

Scikit-learn: Scikit-learn є популярною бібліотекою для машинного навчання, яка підтримує класичні алгоритми, такі як логістична регресія, дерева рішень, випадкові ліси та інші. Вона забезпечує зручний інтерфейс для навчання моделей та їх оцінки.

Бази даних.

Для зберігання та обробки транзакційних даних необхідно використовувати потужні та масштабовані бази даних.

PostgreSQL: PostgreSQL є потужною реляційною базою даних з відкритим кодом, яка підтримує складні запити, транзакції та масштабування. Вона забезпечує високу надійність та продуктивність.

MySQL: MySQL є ще однією популярною реляційною базою даних з відкритим кодом, яка забезпечує високу продуктивність та легкість у використанні. Вона широко використовується у веб-додатках та інших системах.

MongoDB: MongoDB є NoSQL базою даних, яка забезпечує зберігання великих обсягів неструктурованих даних. Вона підтримує масштабування та високу продуктивність при обробці великих даних.

Cassandra: Cassandra є іншою популярною NoSQL базою даних, яка забезпечує високу доступність та масштабування. Вона особливо підходить для розподілених систем та великих обсягів даних.

Інструменти для оркестрації та розгортання.

Для забезпечення масштабованості та керованості системи використовуються інструменти для оркестрації та розгортання контейнерів.

Docker: Docker є платформою для контейнеризації додатків, яка забезпечує їхню портативність та ізоляцію. Використання Docker дозволяє легко розгортати та управляти додатками на різних платформах.

Kubernetes: Kubernetes є платформою для оркестрації контейнерів, яка забезпечує автоматичне масштабування, балансування навантаження та управління розгортанням додатків. Вона є однією з найпопулярніших платформ для управління контейнеризованими додатками.

Інструменти для моніторингу та логування.

Для забезпечення стабільної роботи системи необхідно впровадити інструменти для моніторингу та логування.

Prometheus: Prometheus є системою моніторингу з відкритим кодом, яка забезпечує збір та аналіз метрик продуктивності. Вона підтримує оповіщення та візуалізацію даних.

Grafana: Grafana є платформою для візуалізації даних, яка інтегрується з Prometheus та іншими системами моніторингу. Вона забезпечує створення інтерактивних панелей для візуалізації метрик.

ELK Stack (Elasticsearch, Logstash, Kibana): ELK Stack є потужним інструментом для збору, зберігання та аналізу логів. Elasticsearch забезпечує зберігання та пошук логів, Logstash – збір та обробку логів, а Kibana – візуалізацію даних.

Забезпечення безпеки.

Безпека є критично важливим аспектом для системи виявлення шахрайських транзакцій. Необхідно забезпечити захист даних та системи від несанкціонованого доступу.

TLS (Transport Layer Security): TLS використовується для шифрування даних при передачі між клієнтом та сервером, забезпечуючи захист від прослуховування та підробки даних.

OAuth: OAuth є протоколом авторизації, який забезпечує безпечний доступ до ресурсів без передачі паролів. Він широко використовується для забезпечення безпеки веб-додатків та API.

Регулярні перевірки безпеки: включають перевірку коду на наявність вразливостей, аудит безпеки та тестування на проникнення.

Апаратне забезпечення.

Для забезпечення ефективної роботи системи необхідно використовувати потужні апаратні ресурси.

Центральний процесор (CPU): рекомендується використовувати потужні процесори, такі як Intel Xeon або AMD EPYC, які забезпечують високу продуктивність та багатоядерну обробку.

Графічні процесори (GPU): для прискорення обчислень та навчання моделей глибокого навчання рекомендується використовувати графічні процесори, такі як NVIDIA Tesla або AMD Instinct.

Оперативна пам'ять (RAM): для забезпечення ефективної обробки даних рекомендується використовувати не менше 16 GB RAM.

Твердотільні накопичувачі (SSD): рекомендується використовувати SSD для забезпечення швидкого доступу до даних та прискорення процесів читання/запису.

Фреймворки для веб-розробки.

Для розробки користувацького інтерфейсу та серверної частини використовуються сучасні фреймворки для веб-розробки.

React.js: React.js є популярною бібліотекою для створення користувацьких інтерфейсів, яка забезпечує високу продуктивність та гнучкість завдяки компонентному підходу.

Node.js: Node.js є платформою для серверного програмування, яка забезпечує високопродуктивну обробку запитів завдяки асинхронній моделі вводу/виводу.

Інструменти для резервного копіювання та відновлення.

Для запобігання втратам даних та забезпечення безперервності роботи системи необхідно впровадити механізми резервного копіювання та відновлення.

Bacula: Bacula є системою резервного копіювання з відкритим кодом, яка забезпечує резервне копіювання даних та відновлення у разі аварії.

Сценарії для автоматизації процесів: використання сценаріїв для автоматизації процесів резервного копіювання та відновлення.

Інструменти для візуалізації даних.

Для забезпечення зручності аналізу та прийняття рішень система повинна забезпечувати візуалізацію даних.

D3.js: D3.js є бібліотекою для візуалізації даних, яка забезпечує створення складних та інтерактивних графіків та діаграм.

Chart.js: Chart.js є бібліотекою для створення простих та ефективних графіків та діаграм.

Таким чином, вибір технологій включає детальний аналіз та вибір апаратного та програмного забезпечення, мови програмування, фреймворків, баз даних та інших інструментів, які забезпечать ефективну та надійну роботу системи для виявлення шахрайських транзакцій у блокчейні. Правильний вибір технологій допоможе забезпечити високу продуктивність, масштабованість та безпеку системи.

3.2 Навчання нейронної мережі

Навчання нейронної мережі є одним із ключових етапів розробки системи виявлення шахрайських транзакцій у блокчейні. Процес навчання включає підготовку даних, вибір архітектури моделі, налаштування гіперпараметрів, навчання моделі на тренувальних даних, оцінку її ефективності та оптимізацію. У цьому розділі ми розглянемо детальний процес навчання нейронної мережі, включаючи всі необхідні кроки та аспекти.

Підготовка даних.

Перед початком навчання нейронної мережі необхідно підготувати дані. Це включає збирання, очищення, нормалізацію та розподіл даних на тренувальний, валідаційний та тестовий набори.

Збирання даних: збирання даних про транзакції з блокчейн-мережі, включаючи інформацію про відправника, одержувача, суму, часову мітку та інші атрибути.

Очищення даних: видалення дублікатів, обробка відсутніх значень та аномалій у даних. Очищення даних допомагає забезпечити якість та коректність вхідних даних.

Нормалізація даних: масштабування числових ознак до єдиного діапазону, такого як [0, 1] або [-1, 1], для забезпечення стабільного навчання моделі. Нормалізація допомагає уникнути домінування певних ознак над іншими.

Розподіл даних: розподіл даних на тренувальний (80%), валідаційний (10%) та тестовий (10%) набори. Тренувальний набір використовується для навчання моделі, валідаційний для налаштування гіперпараметрів, а тестовий для оцінки ефективності моделі.

Вибір архітектури моделі.

Архітектура нейронної мережі визначає, як моделі будуть обробляти та аналізувати дані. Для виявлення шахрайських транзакцій у блокчейні

можуть бути використані різні типи нейронних мереж, включаючи рекурентні нейронні мережі (RNN), LSTM, GRU та конволюційні нейронні мережі (CNN).

Рекурентні нейронні мережі (RNN): RNN підходять для аналізу послідовних даних, таких як транзакції у часовому порядку. Вони можуть зберігати інформацію про попередні стани, що дозволяє враховувати історію транзакцій.

LSTM (Long Short-Term Memory): LSTM є спеціальним типом RNN, який здатен зберігати та оновлювати інформацію протягом довгих періодів часу. Вони ефективні для виявлення довготривалих залежностей у даних.

GRU (Gated Recurrent Unit): GRU є спрощеним варіантом LSTM, який має меншу кількість параметрів і може бути більш ефективним для деяких задач.

Конволюційні нейронні мережі (CNN): CNN підходять для вилучення складних ознак з даних та можуть бути ефективними для аналізу графічних представлень транзакцій.

Налаштування гіперпараметрів.

Гіперпараметри визначають структуру та процес навчання нейронної мережі. Їхнє правильне налаштування є критично важливим для досягнення високої продуктивності моделі.

Кількість шарів: визначення кількості шарів у нейронній мережі, включаючи вхідний шар, приховані шари та вихідний шар.

Кількість нейронів: визначення кількості нейронів у кожному шарі. Зазвичай більша кількість нейронів дозволяє моделі навчатися складнішим патернам, але може призвести до перенавчання.

Активаційні функції: вибір активаційних функцій, таких як ReLU (Rectified Linear Unit), Sigmoid, Tanh, які визначають вихід нейронів.

Швидкість навчання (learning rate): визначає, як швидко модель оновлює свої ваги під час навчання. Надто висока швидкість може

призвести до нестабільного навчання, а надто низька — до повільного збіження.

Розмір міні-пакету (batch size): визначає кількість зразків, які обробляються разом під час одного оновлення ваг.

Кількість епох (epochs): визначає кількість проходів через весь тренувальний набір даних.

Навчання моделі.

Процес навчання включає передавання даних через нейронну мережу, обчислення помилки, та оновлення ваг для мінімізації цієї помилки.

Пряме поширення (forward propagation): передавання вхідних даних через шари нейронної мережі для отримання прогнозу.

Обчислення функції втрат: визначення різниці між прогнозованим результатом та фактичним результатом за допомогою функції втрат, такої як середньоквадратична помилка (MSE) або крос-ентропія (cross-entropy).

Зворотне поширення (backpropagation): обчислення градієнтів функції втрат щодо ваг моделі та їхнє оновлення за допомогою методу оптимізації, такого як градієнтний спуск, Adam або RMSprop.

Оцінка ефективності моделі.

Після навчання моделі необхідно оцінити її ефективність на валідаційному та тестовому наборах даних. Це дозволяє визначити, наскільки добре модель узагальнює дані та виявляє шахрайські транзакції.

Метрики оцінки: використання метрик, таких як точність (accuracy), повнота (recall), специфічність (specificity), F1-міра та ROC-AUC для оцінки продуктивності моделі.

Крос-валідація: використання крос-валідації для перевірки стабільності моделі та оцінки її продуктивності на різних підмножинах даних.

Оптимізація моделі.

Після оцінки моделі може виникнути потреба в її оптимізації для покращення продуктивності. Це може включати налаштування

гіперпараметрів, додавання або видалення шарів, зміни активаційних функцій або використання методів регуляризації.

Регуляризація: використання методів регуляризації, таких як L1 та L2 регуляризація, для запобігання перенавчанню моделі.

Dropout: використання техніки dropout, яка випадковим чином відключає деякі нейрони під час навчання, для запобігання перенавчанню.

Техніки збільшення даних (data augmentation): використання технік збільшення даних для створення нових зразків на основі існуючих даних, що допомагає покращити узагальнюючу здатність моделі.

Моніторинг процесу навчання.

Моніторинг процесу навчання допомагає виявити потенційні проблеми та оптимізувати процес у реальному часі.

TensorBoard: використання інструменту TensorBoard для візуалізації процесу навчання, включаючи графіки функції втрат, метрики продуктивності, градієнти та інші показники.

Early Stopping: використання техніки ранньої зупинки (early stopping), яка зупиняє навчання, якщо продуктивність на валідаційному наборі перестає покращуватися, щоб запобігти перенавчанню.

Збереження та завантаження моделі.

Після завершення навчання та оптимізації моделі важливо зберегти її для подальшого використання та завантаження при необхідності.

Збереження моделі: збереження ваг та архітектури моделі у файл для подальшого використання. Це може бути реалізовано за допомогою бібліотек, таких як Keras (метод `model.save()`) або PyTorch (метод `torch.save()`).

Завантаження моделі: завантаження збереженої моделі для використання у прогнозуванні на нових даних або для подальшого навчання. Це може бути реалізовано за допомогою методів `model.load_model()` у Keras або `torch.load()` у PyTorch.

Інтеграція моделі у систему.

Після навчання та збереження моделі необхідно інтегрувати її у систему для виявлення шахрайських транзакцій у реальному часі.

API для прогнозування: розробка API, яке дозволяє іншим компонентам системи взаємодіяти з моделлю для отримання прогнозів. Це може бути реалізовано за допомогою фреймворків, таких як Flask або FastAPI.

Моніторинг продуктивності: постійний моніторинг продуктивності моделі у виробничому середовищі для виявлення потенційних проблем та забезпечення її ефективності.

Таким чином, навчання нейронної мережі включає комплексний процес підготовки даних, вибору архітектури, налаштування гіперпараметрів, навчання, оцінки та оптимізації моделі. Правильне виконання всіх цих кроків забезпечує високу ефективність та надійність системи для виявлення шахрайських транзакцій у блокчейні.

3.3 Опис коду

Опис коду включає детальний розгляд кожного компоненту системи, пояснення логіки та функціоналу коду, а також приклади реалізації ключових частин. У цьому розділі ми розглянемо структуру та реалізацію основних компонентів системи для виявлення шахрайських транзакцій у блокчейні, включаючи збір даних, обробку та збереження даних, навчання та використання моделі машинного навчання, а також інтеграцію та взаємодію між компонентами.

Збір даних.

Перший крок – збір даних про транзакції з блокчейн-мережі. Для цього можна використовувати бібліотеку Web3.py для взаємодії з Ethereum або Bitcoin RPC API для взаємодії з Bitcoin (лістинг 3.1).

Лістинг 3.1 – Збір даних про транзакції з блокчейн-мережі

```
from web3 import Web3

# Підключення до Ethereum вузла
web3 =
Web3(Web3.HTTPProvider('https://mainnet.infura.io/v3/YOUR_INFU
RA_PROJECT_ID'))

# Перевірка підключення
if web3.isConnected():
    print('Connected to Ethereum node')

# Отримання останніх блоків
latest_block = web3.eth.blockNumber
print(f'Latest Ethereum Block: {latest_block}')

# Збір транзакцій з блоку
def get_block_transactions(block_number):
    block = web3.eth.getBlock(block_number,
full_transactions=True)
    transactions = block.transactions
    return transactions

# Приклад отримання транзакцій з останнього блоку
transactions = get_block_transactions(latest_block)
for tx in transactions:
    print(f'Transaction: {tx}')
```

Збереження даних.

Після збору даних необхідно зберегти їх у базі даних. Для цього можна використовувати реляційні бази даних, такі як PostgreSQL, або NoSQL бази даних, такі як MongoDB (лістинг 3.2).

Лістинг 3.2 – Збереження даних

```
import psycopg2

# Підключення до PostgreSQL бази даних
conn = psycopg2.connect(
    dbname="blockchain",
    user="user",
    password="password",
    host="localhost",
    port="5432"
)
cur = conn.cursor()

# Створення таблиці для зберігання транзакцій
```

Продовження лістингу 3.2

```
cur.execute('''
    CREATE TABLE IF NOT EXISTS transactions (
        hash TEXT PRIMARY KEY,
        block_number INTEGER,
        from_address TEXT,
        to_address TEXT,
        value NUMERIC,
        gas INTEGER,
        gas_price NUMERIC,
        input TEXT,
        timestamp TIMESTAMP
    )
''')
conn.commit()

# Збереження транзакцій у базі даних
def save_transaction(tx):
    cur.execute('''
        INSERT INTO transactions (hash, block_number,
from_address, to_address, value, gas, gas_price, input,
timestamp)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
        ON CONFLICT (hash) DO NOTHING
    ''', (tx['hash'], tx['blockNumber'], tx['from'],
tx['to'], web3.fromWei(tx['value'], 'ether'), tx['gas'],
tx['gasPrice'], tx['input'], tx['timestamp']))
    conn.commit()

# Приклад збереження транзакцій
for tx in transactions:
    save_transaction(tx)
```

Обробка даних.

Для навчання моделі необхідно підготувати дані, включаючи нормалізацію та перетворення у формат, зручний для аналізу формат (лістинг 3.3).

Лістинг 3.3 – Підготовка даних

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Завантаження даних з бази даних у DataFrame
cur.execute('SELECT * FROM transactions')
rows = cur.fetchall()
```

Продовження лістингу 3.3

```
    df = pd.DataFrame(rows, columns=['hash', 'block_number',
'from_address', 'to_address', 'value', 'gas', 'gas_price',
'input', 'timestamp'])

    # Нормалізація числових ознак
    scaler = StandardScaler()
    df[['value', 'gas', 'gas_price']] =
scaler.fit_transform(df[['value', 'gas', 'gas_price']])

    # Перетворення часової мітки у числовий формат
    df['timestamp'] =
pd.to_datetime(df['timestamp']).astype(int) / 10**9

    # Розділення даних на тренувальний, валідаційний та
тестовий набори
    train_df = df.sample(frac=0.8, random_state=0)
    test_df = df.drop(train_df.index)
    val_df = test_df.sample(frac=0.5, random_state=0)
    test_df = test_df.drop(val_df.index)

    train_labels = train_df.pop('is_fraud')
    val_labels = val_df.pop('is_fraud')
    test_labels = test_df.pop('is_fraud')
```

Навчання моделі.

Для навчання моделі машинного навчання використовуємо TensorFlow та Keras (лістинг 3.4). Навчання буде виконуватися на підготовлених даних.

Лістинг 3.4 – Навчання моделей

```
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers

    # Побудова моделі
    model = keras.Sequential([
        layers.Dense(64, activation='relu',
input_shape=[len(train_df.keys())]),
        layers.Dropout(0.2),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.2),
        layers.Dense(1, activation='sigmoid')
    ])

    # Компіляція моделі
    model.compile(optimizer='adam',
```

Продовження лістингу 3.4

```
                loss='binary_crossentropy',
                metrics=['accuracy'])

    # Навчання моделі
    history = model.fit(
        train_df, train_labels,
        epochs=50,
        validation_data=(val_df, val_labels),
        verbose=1
    )

    # Оцінка моделі на тестових даних
    test_loss, test_acc = model.evaluate(test_df, test_labels,
verbose=2)
    print(f'Test Accuracy: {test_acc}')
```

Збереження та завантаження моделі.

Після навчання моделі її необхідно зберегти для подальшого використання та завантаження при необхідності (лістинг 3.5).

Лістинг 3.5 – Збереження та завантаження моделі

```
    # Збереження моделі
    model.save('fraud_detection_model.h5')

    # Завантаження моделі
    loaded_model =
keras.models.load_model('fraud_detection_model.h5')
```

Інтеграція моделі у систему.

Розробка API для інтеграції моделі у систему та отримання прогнозів на нових даних (лістинг 3.6).

Лістинг 3.6 – Інтеграція моделі

```
    from flask import Flask, request, jsonify

    app = Flask(__name__)

    # Завантаження моделі
    model =
keras.models.load_model('fraud_detection_model.h5')

    # API для отримання прогнозів
    @app.route('/predict', methods=['POST'])
```

Продовження лістингу 3.6

```
def predict():
    data = request.get_json(force=True)
    df = pd.DataFrame(data)
    predictions = model.predict(df)
    return jsonify({'predictions': predictions.tolist()})

if __name__ == '__main__':
    app.run(debug=True)
```

Моніторинг процесу навчання.

Використання TensorBoard для моніторингу процесу навчання та візуалізації метрик (лістинг 3.7).

Лістинг 3.7 – Моніторинг процесу навчання

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import datetime

# Побудова моделі
model = keras.Sequential([
    layers.Dense(64, activation='relu',
input_shape=[len(train_df.keys())]),
    layers.Dropout(0.2),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(1, activation='sigmoid')
])

# Компіляція моделі
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Налаштування TensorBoard
log_dir = "logs/fit/" +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback =
tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1)

# Навчання моделі з використанням TensorBoard
history = model.fit(
    train_df, train_labels,
    epochs=50,
    validation_data=(val_df, val_labels),
    callbacks=[tensorboard_callback],
```

Продовження лістингу 3.7

```
    verbose=1
)
# Запуск TensorBoard
%load_ext tensorboard
%tensorboard --logdir logs/fit
```

Резервне копіювання та відновлення.

Автоматизація процесу резервного копіювання та відновлення даних та моделей (лістинг 3.8).

Лістинг 3.8 – Резервне копіювання та відновлення

```
#!/bin/bash
# Створення резервної копії бази даних PostgreSQL
pg_dump -U user -h localhost blockchain > backup_$(date
+%F).sql
# Збереження моделі
cp fraud_detection_model.h5
backups/fraud_detection_model_$(date +%F).h5

# Відновлення бази даних
# psql -U user -h localhost blockchain < backup.sql

# Відновлення моделі
# cp backups/fraud_detection_model_$(date +%F).h5
fraud_detection_model.h5
```

Цей детальний опис коду включає всі ключові етапи розробки системи для виявлення шахрайських транзакцій у блокчейні: збір даних, збереження даних, обробка даних, навчання моделі, збереження та завантаження моделі, інтеграція моделі у систему, моніторинг процесу навчання та резервне копіювання та відновлення. Кожен крок детально пояснений з прикладами коду, що допоможе розробникам зрозуміти логіку та реалізувати систему.

3.4 Приклади роботи

Приклади роботи демонструють, як система виявлення шахрайських транзакцій у блокчейні функціонує в реальних умовах. Вони включають

приклади збирання даних, обробки, навчання моделі, прогнозування та реагування на виявлені аномалії. У цьому розділі ми розглянемо різні сценарії використання системи та детально опишемо кожен крок з прикладами коду.

Збирання даних.

Розглянемо приклад збирання даних про транзакції з блокчейн-мережі Ethereum (лістинг 3.9).

Лістинг 3.9 – Збирання даних

```
from web3 import Web3
import json

# Підключення до Ethereum вузла
web3 =
Web3(Web3.HTTPProvider('https://mainnet.infura.io/v3/YOUR_INFU
RA_PROJECT_ID'))

# Функція для збирання транзакцій з блоку
def get_block_transactions(block_number):
    block = web3.eth.getBlock(block_number,
full_transactions=True)
    transactions = block.transactions
    return transactions

# Приклад збирання транзакцій з останнього блоку
latest_block = web3.eth.blockNumber
transactions = get_block_transactions(latest_block)

# Збереження транзакцій у JSON файл
with open('transactions.json', 'w') as f:
    json.dump([dict(tx) for tx in transactions], f,
indent=4)
```

Збереження даних.

Приклад збереження зібраних транзакцій у базу даних PostgreSQL (лістинг 3.10).

Лістинг 3.10 – Збереження даних

```
import psycopg2

# Підключення до PostgreSQL бази даних
conn = psycopg2.connect(
```

Продовження лістингу 3.10

```
    dbname="blockchain",
    user="user",
    password="password",
    host="localhost",
    port="5432"
)
cur = conn.cursor()

# Створення таблиці для зберігання транзакцій
cur.execute('''
    CREATE TABLE IF NOT EXISTS transactions (
        hash TEXT PRIMARY KEY,
        block_number INTEGER,
        from_address TEXT,
        to_address TEXT,
        value NUMERIC,
        gas INTEGER,
        gas_price NUMERIC,
        input TEXT,
        timestamp TIMESTAMP
    )
''')
conn.commit()

# Завантаження транзакцій з JSON файлу
import json
with open('transactions.json', 'r') as f:
    transactions = json.load(f)

# Збереження транзакцій у базі даних
def save_transaction(tx):
    cur.execute('''
        INSERT INTO transactions (hash, block_number,
from_address, to_address, value, gas, gas_price, input,
timestamp)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s,
to_timestamp(%s))
        ON CONFLICT (hash) DO NOTHING
    ''', (tx['hash'], tx['blockNumber'], tx['from'],
tx['to'], Web3.fromWei(int(tx['value']), 'ether'),
int(tx['gas']), Web3.fromWei(int(tx['gasPrice']), 'gwei'),
tx['input'], tx['timestamp']))
    conn.commit()
for tx in transactions:
    save_transaction(tx)
```

Обробка даних.

Приклад обробки даних перед навчанням моделі (лістинг 3.11).

Лістинг 3.11 – Обробка даних

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Завантаження даних з бази даних у DataFrame
cur.execute('SELECT * FROM transactions')
rows = cur.fetchall()
df = pd.DataFrame(rows, columns=['hash', 'block_number',
'from_address', 'to_address', 'value', 'gas', 'gas_price',
'input', 'timestamp'])

# Нормалізація числових ознак
scaler = StandardScaler()
df[['value', 'gas', 'gas_price']] =
scaler.fit_transform(df[['value', 'gas', 'gas_price']])

# Перетворення часової мітки у числовий формат
df['timestamp'] =
pd.to_datetime(df['timestamp']).astype(int) / 10**9

# Розмітка даних (вручну позначимо деякі транзакції як
шахрайські)
df['is_fraud'] = 0
df.loc[df['value'] > 5, 'is_fraud'] = 1  # Наприклад,
позначаємо всі транзакції з сумою більше 5 ETH як шахрайські

# Розділення даних на тренувальний, валідаційний та
тестовий набори
train_df = df.sample(frac=0.8, random_state=0)
test_df = df.drop(train_df.index)
val_df = test_df.sample(frac=0.5, random_state=0)
test_df = test_df.drop(val_df.index)

train_labels = train_df.pop('is_fraud')
val_labels = val_df.pop('is_fraud')
test_labels = test_df.pop('is_fraud')
```

Навчання моделі.

Навчання моделі на підготовлених даних за допомогою TensorFlow та Keras (лістинг 3.12).

Лістинг 3.12 – Навчання моделі

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Побудова моделі
model = keras.Sequential([
    layers.Dense(64, activation='relu',
input_shape=[len(train_df.keys())]),
    layers.Dropout(0.2),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(1, activation='sigmoid')
])

# Компіляція моделі
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Навчання моделі
history = model.fit(
    train_df, train_labels,
    epochs=50,
    validation_data=(val_df, val_labels),
    verbose=1
)

# Оцінка моделі на тестових даних
test_loss, test_acc = model.evaluate(test_df, test_labels,
verbose=2)
print(f'Test Accuracy: {test_acc}')
```

Прогнозування на нових даних.

Приклад прогнозування шахрайських транзакцій на нових даних за допомогою завантаженої моделі (лістинг 3.13).

Лістинг 3.13 – Прогнозування на нових даних

```
# Завантаження моделі
model =
keras.models.load_model('fraud_detection_model.h5')
```

Продовження лістингу 3.13

```
# Приклад нових даних
new_data = pd.DataFrame([{
    'block_number': 1234567,
    'from_address': '0x...',
    'to_address': '0x...',
    'value': scaler.transform([[3]])[0][0],  # 
Масштабування нових даних
    'gas': scaler.transform([[21000]])[0][0],
    'gas_price': scaler.transform([[50]])[0][0],
    'input': '',
    'timestamp': pd.to_datetime('2023-01-01').astype(int)
/ 10**9
}])

# Прогнозування
predictions = model.predict(new_data)
print(f'Predictions: {predictions}')
```

Інтеграція з системою нотифікацій.

Приклад інтеграції з системою нотифікацій для оповіщення про виявлені шахрайські транзакції (лістинг 3.14).

Лістинг 3.14 – Інтеграція з системою нотифікацій

```
from flask import Flask, request, jsonify
import requests

app = Flask(__name__)

# Завантаження моделі
model = 
keras.models.load_model('fraud_detection_model.h5')

# API для отримання прогнозів
@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json(force=True)
    df = pd.DataFrame(data)
    predictions = model.predict(df)

    # Надсилання нотифікацій про шахрайські транзакції
    for i, pred in enumerate(predictions):
        if pred > 0.5:
            tx = data[i]
            notify_fraud(tx)

    return jsonify({'predictions': predictions.tolist()})

def notify_fraud(tx):
```

Продовженя лістингу 3.14

```
        # Приклад нотифікації через електронну пошту
(використовуючи SendGrid API)
        email_data = {
            "personalizations": [{
                "to": [{"email": "admin@example.com"}],
                "subject": "Fraudulent Transaction Detected"
            }],
            "from": {"email": "alert@example.com"},
            "content": [{
                "type": "text/plain",
                "value": f"A fraudulent transaction has been
detected:\n{tx}"
            }]
        }
        headers = {
            'Authorization': 'Bearer YOUR_SENDGRID_API_KEY',
            'Content-Type': 'application/json'
        }
        response =
requests.post('https://api.sendgrid.com/v3/mail/send',
headers=headers, json=email_data)
        return response.status_code

    if __name__ == '__main__':
        app.run(debug=True)
```

Моніторинг процесу навчання.

Використання TensorBoard для моніторингу процесу навчання (лістинг 3.15).

Лістинг 3.15 – Моніторинг процесу навчання

```
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    import datetime

    # Побудова моделі
    model = keras.Sequential([
        layers.Dense(64, activation='relu',
input_shape=[len(train_df.keys())]),
        layers.Dropout(0.2),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.2),
        layers.Dense(1, activation='sigmoid')
    ])

    # Компіляція моделі
```

Продовження лістингу 3.15

```
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    # Налаштування TensorBoard
    log_dir = "logs/fit/" +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback =
tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1)

    # Навчання моделі з використанням TensorBoard
    history = model.fit(
        train_df, train_labels,
        epochs=50,
        validation_data=(val_df, val_labels),
        callbacks=[tensorboard_callback],
        verbose=1
    )

    # Запуск TensorBoard
    %load_ext tensorboard
    %tensorboard --logdir logs/fit
```

Резервне копіювання та відновлення.

Автоматизація процесу резервного копіювання та відновлення даних та моделей (лістинг 3.16).

Лістинг 3.16 – Резервне копіювання та відновлення

```
    #!/bin/bash

    # Створення резервної копії бази даних PostgreSQL
    pg_dump -U user -h localhost blockchain > backup_$(date
+%F).sql

    # Збереження моделі
    cp fraud_detection_model.h5
backups/fraud_detection_model_$(date +%F).h5

    # Відновлення бази даних
    # psql -U user -h localhost blockchain < backup.sql

    # Відновлення моделі
    # cp backups/fraud_detection_model_$(date +%F).h5
fraud_detection_model.h5
```

Ці приклади демонструють, як система виявлення шахрайських транзакцій у блокчейні функціонує на різних етапах: від збору та збереження даних до навчання моделі, прогнозування та інтеграції з системою нотифікацій. Вони включають детальний код та пояснення для кожного кроку, що допоможе розробникам реалізувати та налаштувати систему для їхніх потреб.

3.5 Тестування

Тестування є важливою частиною розробки системи для виявлення шахрайських транзакцій у блокчейні, оскільки воно забезпечує правильність, надійність та ефективність системи. У цьому розділі буде розглянуто різні аспекти тестування, включаючи модульне тестування, інтеграційне тестування, системне тестування, тестування продуктивності та безпеки.

Модульне тестування.

Модульне тестування перевіряє окремі компоненти системи, щоб переконатися, що кожен з них працює правильно. Для Python можна використовувати бібліотеки unittest або pytest (лістинг 3.17).

Лістинг 3.17 – Модульне тестування

```
import unittest
from your_module import get_block_transactions,
save_transaction

class TestBlockchainSystem(unittest.TestCase):

    def test_get_block_transactions(self):
        transactions = get_block_transactions(123456)
        self.assertIsInstance(transactions, list)
        self.assertGreater(len(transactions), 0)

    def test_save_transaction(self):
        tx = {
            'hash': '0x...',
            'blockNumber': 123456,
            'from': '0x...',
            'to': '0x...',
```

Продовження лістингу 3.17

```
            'value': 10,
            'gas': 21000,
            'gasPrice': 50,
            'input': '',
            'timestamp': 1630000000
        }
        result = save_transaction(tx)
        self.assertTrue(result)

if __name__ == '__main__':
    unittest.main()
```

Інтеграційне тестування.

Інтеграційне тестування перевіряє взаємодію між різними компонентами системи, щоб переконатися, що вони працюють разом правильно.

Лістинг 3.18 – Інтеграційне тестування

```
import unittest
import psycopg2
from your_module import get_block_transactions,
save_transaction

class TestIntegration(unittest.TestCase):

    def setUp(self):
        self.conn = psycopg2.connect(
            dbname="blockchain",
            user="user",
            password="password",
            host="localhost",
            port="5432"
        )
        self.cur = self.conn.cursor()

    def tearDown(self):
        self.cur.close()
        self.conn.close()

    def test_transaction_flow(self):
        transactions = get_block_transactions(123456)
        for tx in transactions:
            save_transaction(tx)
        self.cur.execute('SELECT COUNT(*) FROM
transactions WHERE block_number = %s', (123456,))
        count = self.cur.fetchone()[0]
```

Продовження лістингу 3.18

```
        self.assertEqual(count, len(transactions))

if __name__ == '__main__':
    unittest.main()
```

Системне тестування.

Системне тестування перевіряє всю систему як єдине ціле, включаючи всі компоненти та інтеграції. Це включає тестування основних функцій системи, таких як збір даних, збереження даних, навчання моделі, прогнозування та нотифікації (лістинг 3.19).

Лістинг 3.19 – Системне тестування

```
import unittest
import json
from your_module import get_block_transactions,
save_transaction, train_model, predict_fraud, notify_fraud
class TestSystem(unittest.TestCase):
    def test_system_flow(self):
        # Збір даних
        transactions = get_block_transactions(123456)
        self.assertGreater(len(transactions), 0)

        # Збереження даних
        for tx in transactions:
            save_transaction(tx)

        # Навчання моделі
        accuracy = train_model()
        self.assertGreater(accuracy, 0.8)

        # Прогнозування
        with open('new_transactions.json', 'r') as f:
            new_transactions = json.load(f)
        predictions = predict_fraud(new_transactions)
        self.assertIsInstance(predictions, list)

        # Нотифікація
        for i, pred in enumerate(predictions):
            if pred > 0.5:
                tx = new_transactions[i]
                status_code = notify_fraud(tx)
                self.assertEqual(status_code, 202)

if __name__ == '__main__':
    unittest.main()
```

Тестування продуктивності.

Тестування продуктивності перевіряє, як система працює під різними навантаженнями, щоб переконатися, що вона здатна обробляти великі обсяги даних та виконувати операції у межах прийнятного часу (лістинг 3.20).

Лістинг 3.20 – Тестування продуктивності

```
import time
import random
import json
from your_module import save_transaction

def test_performance():
    with open('transactions.json', 'r') as f:
        transactions = json.load(f)

    start_time = time.time()
    for _ in range(1000):  # Тестування на 1000 транзакцій
        tx = random.choice(transactions)
        save_transaction(tx)
    end_time = time.time()
    duration = end_time - start_time
    print(f'Time taken for 1000 transactions: {duration}
seconds')

test_performance()
```

Тестування безпеки.

Тестування безпеки включає перевірку системи на вразливості, такі як SQL-ін'єкції, атаки типу «відмова у обслуговуванні» (DoS), та інші потенційні загрози (лістинг 3.21).

Лістинг 3.21 – Тестування безпеки

```
import requests

def test_sql_injection():
    # Спроба SQL-ін'єкції через API
    response =
requests.post('http://localhost:5000/predict', json={
        'block_number': 123456,
        'from_address': '0x...',
        'to_address': '0x...',
        'value': '1; DROP TABLE transactions;--',
```

Продовження лістингу 3.21

```
            'gas': 21000,
            'gas_price': 50,
            'input': '',
            'timestamp': 1630000000
        })
        assert response.status_code == 400  # Очікується
помилка

    test_sql_injection()
```

Тестування моделі.

Перевірка точності та продуктивності моделі машинного навчання, щоб переконатися, що вона коректно виявляє шахрайські транзакції (лістинг 3.22).

Лістинг 3.22 – Тестування моделі

```
    import unittest
    from your_module import train_model, evaluate_model

    class TestModel(unittest.TestCase):

        def test_model_accuracy(self):
            # Навчання моделі
            model, history = train_model()

            # Оцінка моделі
            accuracy, precision, recall =
evaluate_model(model)
            self.assertGreater(accuracy, 0.8)
            self.assertGreater(precision, 0.75)
            self.assertGreater(recall, 0.75)

    if __name__ == '__main__':
        unittest.main()
```

Тестування нотифікацій.

Перевірка, що система нотифікацій коректно відправляє повідомлення про виявлені шахрайські транзакції.

Лістинг 3.23 – Тестування нотифікацій

```
import unittest
from your_module import notify_fraud
```

Продовження лістингу 3.23

```
class TestNotifications(unittest.TestCase):

    def test_notification(self):
        tx = {
            'hash': '0x...',
            'blockNumber': 123456,
            'from': '0x...',
            'to': '0x...',
            'value': 10,
            'gas': 21000,
            'gasPrice': 50,
            'input': '',
            'timestamp': 1630000000
        }
        status_code = notify_fraud(tx)
        self.assertEqual(status_code, 202)  # Перевірка
успішного відправлення

if __name__ == '__main__':
    unittest.main()
```

Регресійне тестування.

Регресійне тестування перевіряє, що зміни в коді не порушили існуючу функціональність системи. Це важливо після внесення будь-яких змін або оновлень у систему.

Лістинг 3.24 – Регресійне тестування

```
import unittest

class TestRegression(unittest.TestCase):

    def test_existing_features(self):
        # Повторне тестування існуючих функцій після
оновлень
        self.test_get_block_transactions()
        self.test_save_transaction()
        self.test_system_flow()

    def test_get_block_transactions(self):
        transactions = get_block_transactions(123456)
        self.assertIsInstance(transactions, list)
        self.assertGreater(len(transactions), 0)

    def test_save_transaction(self):
        tx = {
            'hash': '0x...',
```

Продовження лістингу 3.24

```
            'blockNumber': 123456,
            'from': '0x...',
            'to': '0x...',
            'value': 10,
            'gas': 21000,
            'gasPrice': 50,
            'input': '',
            'timestamp': 1630000000
        }
        result = save_transaction(tx)
        self.assertTrue(result)

    def test_system_flow(self):
        # Збір даних
        transactions = get_block_transactions(123456)
        self.assertGreater(len(transactions), 0)

        # Збереження даних
        for tx in transactions:
            save_transaction(tx)

        # Навчання моделі
        accuracy = train_model()
        self.assertGreater(accuracy, 0.8)

        # Прогнозування
        with open('new_transactions.json', 'r') as f:
            new_transactions = json.load(f)
        predictions = predict_fraud(new_transactions)
        self.assertIsInstance(predictions, list)

        # Нотифікація
        for i,

 pred in enumerate(predictions):
            if pred > 0.5:
                tx = new_transactions[i]
                status_code = notify_fraud(tx)
                self.assertEqual(status_code, 202)

if __name__ == '__main__':
    unittest.main()
```

Документування тестів.

Документування тестів включає створення звітів про результати тестування для подальшого аналізу та оптимізації системи.

```
# Запуск тестів та створення звіту
pytest --junitxml=report.xml
```

Таким чином, тестування системи для виявлення шахрайських транзакцій у блокчейні охоплює широкий спектр перевірок, включаючи модульне, інтеграційне, системне, продуктивне та безпекове тестування. Кожен тип тестування допомагає забезпечити коректну роботу системи, її надійність та ефективність, а також захист від можливих вразливостей.

ВИСНОВКИ

Розробка системи для виявлення шахрайських транзакцій у блокчейні є складним, але важливим завданням, яке вимагає застосування сучасних методів машинного та глибокого навчання, ефективних алгоритмів обробки даних та надійних технологій для забезпечення безпеки та продуктивності. У цьому розділі ми підсумуємо результати роботи, розглянемо ключові аспекти розробки та їхню значущість, а також окреслимо можливі напрями подальшого розвитку системи.

Збір даних про транзакції з блокчейн-мережі є основою для подальшого аналізу та виявлення шахрайства. Використання сучасних інструментів, таких як Web3.py для взаємодії з Ethereum або Bitcoin RPC API для роботи з Bitcoin, дозволяє автоматизувати процес збору даних та забезпечити їх актуальність.

Підготовка даних включає очищення, нормалізацію та перетворення даних у формат, зручний для аналізу. Це дозволяє забезпечити високу якість вхідних даних, що є критично важливим для ефективності моделей машинного навчання.

Для виявлення шахрайських транзакцій були розглянуті різні моделі машинного та глибокого навчання, такі як логістична регресія, дерева рішень, випадкові ліси, рекурентні нейронні мережі (RNN), LSTM та конволюційні нейронні мережі (CNN). Кожна з цих моделей має свої переваги та недоліки, і вибір конкретної моделі залежить від специфіки завдання та типу даних.

Навчання моделей включає налаштування гіперпараметрів, навчання на тренувальних даних, оцінку на валідаційних даних та тестування на тестових даних. Правильне налаштування моделей дозволяє досягти високої точності та ефективності виявлення шахрайства.

Інтеграція моделей машинного навчання у систему для виявлення шахрайських транзакцій включає розробку API для прогнозування,

моніторинг у реальному часі, систему нотифікацій та автоматичне створення звітів. Використання таких інструментів, як Flask для створення API та SendGrid для нотифікацій, забезпечує зручність та оперативність роботи системи.

Тестування є важливою частиною розробки системи, яка включає модульне, інтеграційне, системне, продуктивне та безпекове тестування. Кожен тип тестування допомагає виявити потенційні проблеми та забезпечити стабільну та надійну роботу системи.

Оптимізація системи включає налаштування гіперпараметрів моделей, використання методів регуляризації, таких як L1 та L2, а також технік збільшення даних для покращення узагальнюючої здатності моделей.

Безпека є критично важливим аспектом для системи виявлення шахрайських транзакцій. Використання методів шифрування, таких як TLS, та протоколів авторизації, таких як OAuth, забезпечує захист даних та системи від несанкціонованого доступу. Регулярні перевірки безпеки та аудит системи допомагають виявити та виправити вразливості.

Масштабованість системи забезпечується за допомогою використання таких інструментів, як Docker для контейнеризації додатків та Kubernetes для оркестрації контейнерів. Це дозволяє автоматично масштабувати ресурси та забезпечувати високу продуктивність системи при збільшенні навантаження.

Тестування продуктивності допомагає оцінити, як система працює під різними навантаженнями, та забезпечити її здатність обробляти великі обсяги даних у межах прийнятного часу.

Ретельне навчання та тестування моделі забезпечують високу точність та надійність виявлення шахрайських транзакцій. Використання метрик, таких як точність, повнота, специфічність, F1-міра та ROC-AUC, допомагає оцінити ефективність моделі та її здатність виявляти шахрайство.

Автоматизація процесів, таких як збирання даних, навчання моделей, прогнозування та нотифікації, забезпечує ефективність роботи системи та

зменшує кількість ручної роботи. Це дозволяє швидше виявляти шахрайські транзакції та оперативно реагувати на загрози.

Документування всіх етапів розробки та тестування системи є важливим для забезпечення її підтримки та подальшого розвитку. Створення детальної документації дозволяє іншим розробникам легко зрозуміти логіку системи та внести необхідні зміни або покращення.

Система для виявлення шахрайських транзакцій у блокчейні має великий потенціал для подальшого розвитку. Це включає:

– покращення моделей машинного навчання за допомогою використання нових методів та алгоритмів;

– інтеграція з новими блокчейн-платформами для забезпечення ширшого охоплення;

– розробка нових функцій, таких як аналіз поведінкових патернів користувачів та виявлення більш складних типів шахрайства;

– оптимізація продуктивності системи для роботи з великими обсягами даних у реальному часі.

Розробка та впровадження системи для виявлення шахрайських транзакцій у блокчейні є складним завданням, яке вимагає застосування сучасних технологій та методів. Однак, правильно реалізована система може значно знизити ризики шахрайства та забезпечити безпеку фінансових операцій у блокчейні. Дослідження методів класифікації шахрайських транзакцій, вибір та налаштування моделей машинного навчання, інтеграція системи, тестування та оптимізація – все це є ключовими кроками для досягнення успішного результату. Подальший розвиток системи дозволить ще більше підвищити її ефективність та надійність, що є критично важливим у сучасному світі цифрових фінансів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. URL: https://bitcoin.org/bitcoin.pdf (дата звернення: 02.05.2024).

2. Cachin C. Architecture of the Hyperledger Blockchain Fabric. *Proceedings of the Workshop on Distributed Cryptocurrencies and Consensus Ledgers*. 2016. URL: https://arxiv.org/abs/1602.06418 (date of access: 02.05.2024).

3. Wood G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper* 2014. URL: https://ethereum.github.io/yellowpaper/paper.pdf (дата звернення: 02.05.2024).

4. Antonopoulos A. Mastering Bitcoin: Unlocking Digital Cryptocurrencies, 2nd ed. O'Reilly Media, 2017.

5. Antonopoulos A., Wood G. Mastering Ethereum: Building Smart Contracts and DApps. O'Reilly Media, 2018.

6. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016.

7. Haykin S., Neural Networks and Learning Machines, 3rd ed. Pearson, 2008.

8. Web3.py Documentation. URL: https://web3py.readthedocs.io/en/stable/ (date of access: 02.05.2024).

9. Developer Guides – Bitcoin. *Getting Started – Bitcoin*. URL: https://developer.bitcoin.org/devguide/ (date of access: 02.05.2024).

10. PostgreSQL: Documentation. *PostgreSQL: The world's most advanced open source database*. URL: https://www.postgresql.org/docs/ (date of access: 02.05.2024).

11. Guide | TensorFlow Core. *TensorFlow*. URL: https://www.tensorflow.org/guide (date of access: 02.05.2024).

12. Keras: Deep Learning for humans. *Keras: Deep Learning for humans*. URL: https://keras.io/ (date of access: 02.05.2024).

13. Home | ethereum.org. *ethereum.org*. URL: https://ethereum.org/ (date of access: 02.05.2024).

14. Bitcoin - Open source P2P money. *Bitcoin - Open source P2P money*. URL: https://bitcoin.org/ (date of access: 02.05.2024).