НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Інформаційно-аналітична система графового типу для прийняття рішень в бізнесі»

Виконав:
студент IV курсу, групи КІ-02
Наход Олексій Олександрович _____

Керівник:
проф. кафедри ШІ, д.т.н., професор,
Данилов Валерій Якович _____

Консультант з економічного розділу:
проф. кафедри ЕК ФММ, д.е.н., професор,
Шевчук Олена Анатоліївна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ,
Кравець П. В. _____

Рецензент:
директор НН ФТІ, д.т.н., професор,
Новіков Олексій Миколайович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«31» січня 2024 р.

ЗАВДАННЯ

на дипломну роботу студенту

Находу Олексію Олександровичу

1. Тема роботи «Інформаційно-аналітична система графового типу для прийняття рішень в бізнесі», керівник роботи Данилов Валерій Якович, професор, д.т.н., затверджені наказом по університету від «31» травня 2024 р. № 2240-С

2. Термін подання студентом роботи «10» червня 2024 року.

3. Вихідні дані до роботи: різнотипна бізнес-інформація (компанії, клієнти, товари) із загальнодоступних наборів даних.

4. Перелік завдань:

1. Дослідити актуальність використання графових систем у бізнесі.
2. Здійснити огляд методів навчання графових нейронних мереж.
3. Зібрати бізнес-дані з відкритих джерел для їх подальшого аналізу.
4. Розробити систему прогнозування з використанням графових баз даних та нейронних мереж.
5. Провести аналіз даних за допомогою розробленої системи.
6. Розглянути альтернативні архітектури розробленої системи та порівняти результати.

7. Зробити висновки щодо точності системи, проаналізувати можливості її інтеграції в процес ухвалення рішень компаніями, визначити напрямок для подальших досліджень.

5. Ілюстративний матеріал: слайди презентації, виконані в середовищі Microsoft PowerPoint.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук О. А., професор		

7. Дата видачі завдання «05» лютого 2024 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формулювання тематики дослідження	22.04.2024	Виконано
2	Аналіз літературних джерел	29.04.2024	Виконано
3	Математичне моделювання задачі	06.05.2024	Виконано
4	Розробка програмного продукту	13.05.2024	Виконано
5	Тестування програмного продукту	20.05.2024	Виконано
6	Експериментальна перевірка системи	27.05.2024	Виконано
7	Аналіз отриманих результатів	03.06.2024	Виконано
8	Оформлення розділів дипломної роботи	10.06.2024	Виконано

Студент Олексій НАХОД

Керівник Валерій ДАНИЛОВ

РЕФЕРАТ

Дипломна робота: 162 с., 40 рис., 19 табл., 45 посилань, 1 додаток.

ГРАФОВА БАЗА ДАНИХ, ГРАФОВА НЕЙРОННА МЕРЕЖА, ГРАФ ЗНАНЬ, ВЕЛИКА МОВНА МОДЕЛЬ, СИСТЕМА ПІДТРИМКИ РІШЕНЬ, БІЗНЕС-АНАЛІТИКА

Об’єкт дослідження: процеси обробки та аналізу даних для ухвалення рішень у компаніях.

Предмет дослідження: інформаційно-аналітична система графового типу для підтримки ухвалення бізнес-рішень на основі різнотипних даних.

Мета роботи: розробка системи збору, очищення, інтеграції та аналізу різнорідних бізнес-даних із застосуванням графової бази даних, графової нейронної мережі та великої мовної моделі.

Актуальність роботи полягає у необхідності пошуку альтернативних підходів для аналізу великих обсягів різнорідної інформації з метою ухвалення обґрунтованих бізнес-рішень. Традиційні реляційні бази даних мають обмежену здатність для проведення такого аналізу; натомість графові бази даних у поєднанні з методами машинного навчання на графах та великими мовними моделями надають можливість комплексної обробки інформації та генерування змістовних рекомендацій.

Результати роботи: виконано огляд переваг графових систем для інтелектуального аналізу бізнес-даних; розглянуто основні методи розв’язання задач аналізу даних за допомогою графів; запропоновано архітектуру графової інформаційної системи, що поєднує графову базу даних, компоненти розпізнавання зображень та аудіо, графову нейронну мережу та велику мовну модель; проаналізовано потенційні можливості для інтеграції розробленої системи у бізнес-процеси компаній.

Основні положення дослідження доповідалися на двох міжнародних наукових конференціях та опубліковано у двох фахових виданнях категорії Б.

ABSTRACT

Bachelor's thesis: 162 p., 40 figures, 19 tables, 45 references, 1 appendix.

GRAPH DATABASE, GRAPH NEURAL NETWORK, KNOWLEDGE GRAPH, LARGE LANGUAGE MODEL, DECISION SUPPORT SYSTEM, BUSINESS ANALYTICS

Object of research: data processing and analysis for decision-making in companies.

Subject of research: a graph-based information and analytics system to support business decision-making based on heterogeneous data.

Purpose of research: to develop a system for collecting, cleaning, integrating, and analyzing heterogeneous business data using a graph database, a graph neural network, and a large language model.

The relevance of the work lies in the need to find alternative approaches for analyzing large volumes of heterogeneous information in order to make informed business decisions. Traditional relational databases have limited capacity for such analysis; instead, graph databases combined with graph machine learning methods and large language models provide the capability for comprehensive information processing and generation of meaningful recommendations.

Results: an overview of the advantages of graph systems for business data mining is provided; the main methods of solving data mining problems using graphs are considered; the architecture of a graph information system combining a graph database, image and audio recognition components, a graph neural network, and a large language model is proposed; potential opportunities for integrating the developed system into business processes of companies are analyzed.

The main provisions of the research were presented at two international scientific conferences and published in two B-category professional journals.

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1 ФУНДАМЕНТАЛЬНІ ПРИНЦИПИ СИСТЕМ ГРАФОВОГО ТИПУ В ГАЛУЗІ БІЗНЕС-АНАЛІТИКИ 10
1.1 Базові положення теорії графів для обробки бізнес-даних 10
1.2 Рівні аналізу графів у бізнес-системах 13
1.3 Приклади використання графових систем для розв’язання задач бізнесу 20
1.4 Постановка задачі 22
Висновки до розділу 1 22
РОЗДІЛ 2 ЗАСТОСУВАННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ АНАЛІЗУ ГРАФОВИХ СТРУКТУР У БІЗНЕСІ 24
2.1 Огляд алгоритмічних підходів для побудови векторних представлень графових бізнес-даних 24
2.2 Бізнес-орієнтовані архітектури GNN 33
2.3 Методики навчання GNN для задач прогнозування в бізнесі 42
2.4 Імплементація GNN на графових наборах бізнес-даних 48
Висновки до розділу 2 55
РОЗДІЛ 3 РОЗРОБКА ГРАФОВОЇ СИСТЕМИ ПІДТРИМКИ БІЗНЕС-РІШЕНЬ57
3.1 Архітектура розробленої системи підтримки бізнес-рішень 58
3.2 Створення графових нейронних мереж для прогнозування бізнес-даних 62
3.3 Взаємодія з розробленою системою через API 74
3.4 Взаємодія з розробленою системою через інтерфейс користувача. 78
Висновки до розділу 3 82
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ГРАФОВОЇ СИСТЕМИ ПІДТРИМКИ БІЗНЕС-РІШЕНЬ 83
4.1 Постановка задачі проєктування 83

4.2 Обґрунтування функцій програмного продукту 84
4.3 Обґрунтування системи параметрів програмного продукту..... 87
4.4 Аналіз експертного оцінювання параметрів..... 90
4.5 Аналіз рівня якості варіантів реалізації функцій 94
4.6 Економічний аналіз варіантів розробки ПП 95
4.7 Вибір кращого варіанта ПП техніко-економічного рівня 102
Висновки до розділу 4 104
ВИСНОВКИ..... 105
ПЕРЕЛІК ПОСИЛАНЬ..... 107
ДОДАТОК А ЛІСТИНГ ПРОГРАМИ 113

ВСТУП

Сучасний бізнес розглядає дані як нову валюту, бо вміння перетворювати безформні потоки інформації на зважені рішення стало потужним інструментом у руках прогресивних компаній, надаючи їм беззаперечну перевагу над конкурентами, які все ще покладаються на застарілі методи та інтуїцію. Традиційні системи управління даними часто не справляються з обробкою складних зв'язків та відображенням корисної аналітики в режимі реального часу. Це спонукає до пошуку більш досконалих технологій, здатних ефективно моделювати та видобувати взаємопов'язані дані для вдосконалення процесів ухвалення рішень.

Графові бази даних забезпечують надійний фундамент для розробки сучасних аналітичних систем, оскільки їм притаманна здатність ілюструвати взаємозв'язки між різнотипною інформацією. Графова парадигма надає можливість управління складними мережами зв'язків, що робить її пристосованою до використання у системах бізнес-аналітики та підтримки ухвалення рішень. У першому розділі розглянуті переваги графової парадигми для побудови інформаційно-аналітичної системи, що надає глибше розуміння даних, якими володіє бізнес.

Поява графових нейронних мереж та адаптація раніше розроблених згорткових мереж та мереж-трансформерів на графи дозволили застосувати передові методи машинного навчання для аналізу даних, представлених у вигляді графів. У другому розділі розглянуто архітектури графових мереж, процес їх навчання, можливі модифікації та способи прогнозування на основі інформації, що зберігається у графовій базі даних.

Для інтеграції провідних компонентів на зразок графових баз даних та нейронних мереж доцільним є вибір сучасних технологій розробки програмного забезпечення. У третьому розділі висвітлено процес створення вебзастосунку за допомогою мови програмування Python з фреймворком Flask.

Описано архітектуру системи, що поєднує графову базу даних, графову нейронну мережу та велику мовну модель для надання точних відповідей на запит користувача. Продемонстровано процес навчання нейронних мереж на класичних наборах даних для надання подальших прогнозів. Зображено побудову структурованого API, що дозволяє отримати доступ до системи на програмному рівні, та інтуїтивного інтерфейсу, що надає доступ до системи через браузер.

Поява великих мовних моделей спричинила стрімку генерацію нових знань, які допомагають ухвалювати стратегічні бізнес-рішення. Використання потужної LLM в розробленій системі розширює її функціональність, збільшуючи правдивість згенерованих даних за допомогою надання доступу до внутрішньої інформації організації.

Ключовою особливістю інформації, яка зберігається в базах даних компаній, є їхня різноманітність – від звичайної текстової фінансової звітності до складніших неструктурованих даних, наприклад, зображень та аудіо. Стрімкий розвиток сфер обробки природної мови та машинного навчання надав можливість отримати цінну інформацію з таких форматів даних. У роботі продемонстровано компоненти image-to-text та audio-to-text, що дозволяють перетворити різнотипні дані до текстового формату для полегшення проведення подальшого аналізу.

Для оцінки економічної доцільності перетворення створеного прототипу на комерційний застосунок необхідно порівняти можливі стратегії втілення компонентів системи. У четвертому розділі виконано функціонально-вартісний аналіз розробленої системи з метою мінімізації затрат на стадіях проєктування, виробництва й експлуатації зі збереженням функціональності та задоволенням потреб користувачів.

РОЗДІЛ 1 ФУНДАМЕНТАЛЬНІ ПРИНЦИПИ СИСТЕМ ГРАФОВОГО ТИПУ В ГАЛУЗІ БІЗНЕС-АНАЛІТИКИ

1.1 Базові положення теорії графів для обробки бізнес-даних

У сучасному світі дані стають все складнішими та більш взаємопов'язаними. Через це з'явилася потреба у нових підходах до зберігання та опрацювання даних, одним з яких є графові бази даних.

Граф $G(V, E)$ – це абстрактний тип даних, що складається з множини *вершин* (або *вузлів*, *точок*) $V = \{v_1, v_2, \dots, v_n\}$ та множини пар цих вершин $E = \{(v_i, v_j) | v_i, v_j \in V\}$, що називаються *ребрами* (або *зв'язками*, *відрізками*) [1].

Графи поділяються на: *орієнтовані* (зв'язки мають напрям) та *неорієнтовані* (зв'язки не мають напряму), *зважені* (зв'язки мають індивідуальну вагу) та *незважені* (зв'язки мають однакову вагу), *зв'язні* (між будь-якими вершинами існує маршрут) та *незв'язні* (існує пара вершин, між якими нема маршруту), також існують *мультиграфи* (деякі ребра мають одні й ті ж кінцеві вершини) [2].

Графи є загальним способом представлення об'єктів, між якими існують зв'язки. Велика кількість систем може бути зображена у вигляді графів: комп'ютерні мережі, хімічні молекули, транспортна інфраструктура, нейрони в мозку тощо [3]. Наприклад, маємо перелік технологічних компаній – Apple, Google, Microsoft, Amazon – і такі зв'язки між ними:

- Apple співпрацює з Google (наприклад, у сфері картографічних даних);
- Google конкурує з Microsoft (наприклад, у хмарних обчисленнях);
- Microsoft надає Amazon послуги (наприклад, через партнерство з Amazon Web Services);
- Amazon співпрацює з Apple (наприклад, продає продукти Apple);
- Apple конкурує з Microsoft (наприклад, в операційних системах та

пристроях).

Ці дані можна представити у вигляді орієнтованого графа (рис. 1.1):

$$V = \{v_1 = Apple, v_2 = Google, v_3 = Microsoft, v_4 = Amazon\},$$
$$E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1), (v_1, v_3)\}.$$

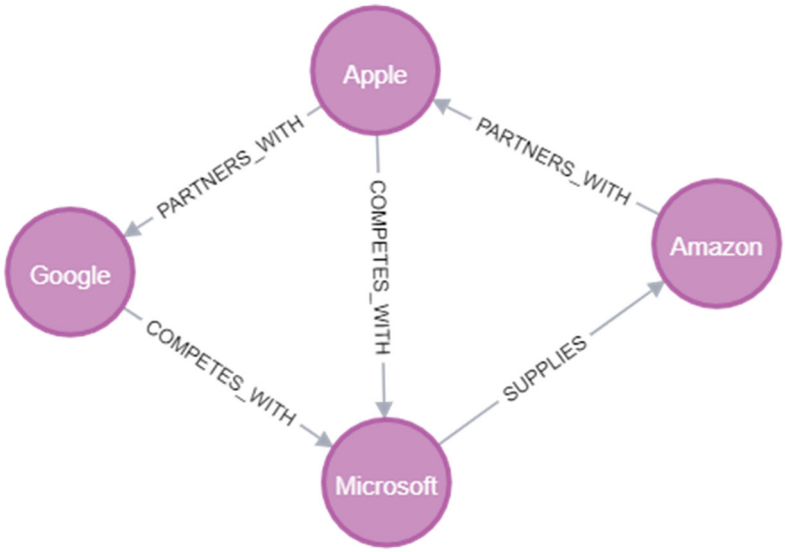

Рисунок 1.1 – Зв'язки між технологічними компаніями

Степінь вершини v_i неорієнтованого графа – кількість ребер, суміжних до цієї вершини, позначається $\deg(v_i)$. *Степінь входу вершини* v_i орієнтованого графа – кількість ребер, що входять у вершину, позначається $\deg^{in}(v_i)$. *Степінь виходу вершини* v_i орієнтованого графа – кількість ребер, що виходять з вершини, позначається $\deg^{out}(v_i)$ [2].

Біграф – граф, множину вузлів якого можна розділити на дві незалежні неперетинні підмножини U та V, тобто кожне ребро з'єднує вузол з підмножини U та вузол з підмножини V, а U та V не мають спільних елементів.

Графи можуть бути представлені двома основними способами [3].

1 Матриця суміжності A: $A_{ij} = 1$, якщо існує ребро, що йде з вершини v_i у вершину v_j; $A_{ij} = 0$ в іншому випадку. Наприклад,

матрицю суміжності з рис. 1.1 можна представити в такому вигляді:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

За допомогою матриці суміжності можна знайти степінь вершини $\deg(v_i) = \sum_{j=1}^{N} A_{ij}$. Більшість мереж у реальному світі мають розріджену матрицю A_{ij}.

2 Список суміжності: кожній вершині v_i ставиться у відповідність список, що містить усі вершини v_j такі, що $(v_i, v_j) \in$ E. Список суміжності дозволяє легше працювати з великими розрідженими мережами.

Сучасне глибинне навчання сфокусоване на простіших типах даних, а саме на послідовностях (текст або звук) та сітках (зображення). Однак більшість реальних явищ, що представляються у вигляді мереж, не піддаються такій обробці через їхню складність [4]. Приклади проблем, що виникають при застосуванні класичних нейронних мереж для аналізу бізнес-даних:

- довільний розмір та складна топологія: глобальна мережа ланцюгів постачальників, виробників, дистриб'юторів та клієнтів утворює комплексну екосистему з багатьма рівнями зв'язків;
- відсутність просторової локальності: у тексті можна визначити напрями $\{left, right\}$, в зображеннях – $\{up, down, left, right\}$, а в мережах не існує певної точки відліку і напрямків;
- відсутність встановленого порядку елементів: об'єкти мережі співавторства в наукових дослідженнях фармацевтичних компаній не можуть бути інтуїтивно пронумеровані;
- високий ступінь динамічності: користувачі соціальних мереж (Facebook, Twitter) та зв'язки між ними постійно створюються,

змінюються, видаляються;

- мультимодальність: платформи електронної комерції (Amazon, Alibaba) зберігають не тільки текстовий опис товарів, а і зображення, відеоогляди користувачів тощо.

Через такі особливості потрібен спеціальний підхід для створення та застосування нейронних мереж на графах.

1.2 Рівні аналізу графів у бізнес-системах

Задачі, що розглядаються при роботі з бізнес-графами, мають чотири рівні [1].

1. Рівень вузлів: система управління запасами ідентифікує товар з низьким рівнем продажів як потенційний кандидат на виведення з асортименту.
2. Рівень зв’язків: система рекомендацій в інтернет-магазині пропонує клієнту придбати певний товар на основі його попередніх покупок та схожості з іншими товарами.
3. Рівень спільнот: алгоритм сегментації клієнтів у CRM-системі виявляє групу цінних замовників зі схожими характеристиками та поведінкою для персоналізованих маркетингових кампаній.
4. Рівень графа: система виявлення шахрайства в банку визначає, що певний шаблон транзакцій вказує на високий ризик неавторизованого використання банківської картки.

Рівень вузлів

Тип задач на рівні вузлів

Маємо задачу класифікації: дано множину вузлів з мітками (англ. *labels*), потрібно поставити у відповідність мітки іншим вузлам [4].

Характеристики вузлів

– степінь: підраховує кількість суміжних вузлів:

$$\deg(v_i) = |N(v_i)|,$$

де $N(v_i)$ – множина суміжних до v_i вузлів;

– центральність: враховує важливість вузла в графі

- на основі власних векторів (англ. *eigenvector centrality*):

$$c_v = \frac{1}{\lambda}\sum_{u \in N(v)} c_u,$$

де λ – власне значення, пов'язане з вектором центральності;

- на основі степеня посередництва (англ. *betweenness centrality*):

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

де σ_{st} – кількість найкоротших шляхів між s та t, $\sigma_{st}(v)$ – кількість таких шляхів, що проходять через v;

- на основі степеня близькості (англ. *closeness centrality*):

$$c_v = \frac{1}{\sum_{u \neq v} d(u,v)},$$

де $d(u,v)$ – довжина найкоротшого шляху між u та v;

– коефіцієнт кластеризації:

$$e_v = \frac{E(N(v))}{\binom{k_v}{2}} \in [0,1],$$

де $E(N(v))$ – кількість ребер між сусідами вузла v;

k_v – кількість суміжних вузлів;

- вектор степеня графлетів (англ. *graphlet degree vector*): оцінка топології локальної мережі вузла.

Графлети g_i – підграфи, яким притаманні:

- неізоморфність: не можна перейменувати вершини графа, щоб зробити його ідентичним іншому графу;
- породженість: підграф утворений з підмножини вершин графа разом з усіма ребрами, що з'єднують пари вершин з цієї підмножини.

Спосіб розв'язання задач на рівні вузлів

Для розв'язання таких задач знаходять оцінки $c(v)$ для кожного вузла v та передбачають мітки для цих вузлів на основі їхніх векторів ознак.

Рівень зв'язків

Типи задач на рівні зв'язків

- приховують випадкові зв'язки та намагаються їх передбачити;
- дана поточна множина зв'язків, намагаються передбачити появу нових зв'язків з часом.

Характеристики зв'язків

- відстань: довжина найкоротшого шляху між двома вузлами v_i та v_j;
- перекриття локального сусідства (англ. *local neighborhood overlap*): враховує кількість вузлів, що належать і $N(v_i)$, і $N(v_j)$;
    - спільні сусіди:

$$c(v_i, v_j) = \left|N(v_i) \cap N(v_j)\right|;$$

- коефіцієнт Жаккара:

$$c(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|};$$

- індекс Адамік–Адар:

$$c(v_i, v_j) = \sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log k_u};$$

– перекриття глобального сусідства (англ. *global neighborhood overlap*)

- індекс Катца: підраховує кількість шляхів усіх довжин між парою вузлів (v_i, v_j):

$$c(v_i, v_j) = \sum_{l=1}^{\infty} \beta^l A^l_{v_1 v_2},$$

де $0 < \beta < 1$ – коефіцієнт знецінювання,
$A^l_{v_1 v_2}$ – матриця суміжності між v_i та v_j, що містить кількість шляхів довжин l.

Спосіб розв'язання задач на рівні зв'язків

Для розв'язання таких задач знаходять оцінки $c(v_i, v_j)$ для кожної пари вузлів (v_i, v_j) та передбачають ймовірність існування зв'язків між парами вузлів за їхнім вектором ознак [2]. Пари з найвищими ймовірностями вважаються потенційними новими зв'язками.

Рівень спільнот

Задачі на рівні спільнот зосереджені на виявленні груп вузлів зі схожими характеристиками або щільними зв'язками між собою. Значний інтерес до цих задач зумовлений дослідженнями, що підтвердили існування таких принципів у

реальних бізнес-мережах [5]:

- між вузлами існують *короткі зв'язки*, що об'єднують вузли однієї підмережі, та *довгі зв'язки*, що об'єднують різні частини мережі;
- короткі зв'язки є сильними, довгі зв'язки є слабкими;
- короткі зв'язки дублюють більшість інформації, довгі зв'язки надають нову інформацію;
- *тріадне замикання* (англ. *triadic closure*): якщо існують зв'язки (A, B) і (B, C), то з великою ймовірністю з'явиться зв'язок (A, C).

Тип задач на рівні спільнот

Виявлення спільнот (англ. *community detection*): пошук груп вузлів, які мають більше зв'язків між собою, ніж з рештою графа. Спільноти можуть перетинатися, тобто вузол може належати до кількох спільнот одночасно.

Характеристики спільнот

- модулярність Q: міра якості розбиття графа на спільноти, порівнює щільність зв'язків усередині спільнот з очікуваною щільністю зв'язків у випадковому графі з такою ж послідовністю степенів вузлів:

$$Q(G,S) = \frac{1}{2m}\sum_{s\in S}\sum_{i\in s}\sum_{j\in s}(A_{ij} - \frac{k_i k_j}{2m}),$$

де m – кількість ребер;

k_i – степінь вузла i;

- перетин зв'язків:

$$O_{ij} = \frac{|(N(i)\cap N(j))-\{i,j\}|}{|(N(i)\cup N(j))-\{i,j\}|},$$

де $N(i)$ – сусіди вузла i.

Спосіб розв'язання задач на рівні спільнот

Алгоритм Louvain [6] використовує жадібну максимізацію модулярності для виявлення ієрархічних спільнот.

1. Для кожного вузла i обчислюють приріст модулярності $\nabla Q(D \to i \to C)$ переходу вузла i зі спільноти D у спільноту C та переміщують вузол i у спільноту з найбільшим приростом модулярності. Цей процес повторюють, доки приріст модулярності не зупиниться.
2. Отримані спільноти комбінують у вузли, причому зв'язки між ними існують, якщо існували зв'язки між вузлами початкових спільнот, а вага зв'язку дорівнює сумі ваг між відповідними спільнотами. Якщо можливо покращити модулярність, переходять до кроку 1, в іншому випадку алгоритм завершується.

Обчислювальна складність: $O(n \log n)$, де n – кількість вузлів у мережі. Завдяки високій ефективності, алгоритм Louvain можна застосовувати до бізнес-графів великих розмірів, наприклад, для виявлення схожих груп користувачів у соціальних мережах.

Рівень графа

Тип задач на рівні графа

Основною задачею є класифікація: на основі характеристик графа потрібно поставити у відповідність правильну мітку [2].

Характеристики графа

1. Ядро графлетів K оцінює кількість графлетів у графі, використовуючи допоміжну функцію f_G:

$$(f_G)_i = |\{g_i \subseteq G\}|, i = 1,2, \dots, n_k,$$

де n – розмір графа;

k – розмір графлета.

Щоб мати можливість порівнювати графи різних розмірів, проводиться нормалізація функції f_G:

$$(h_G)_i = \frac{(f_G)_i}{\sum_i (f_G)_i},$$
$$K(G, G') = h_G^T h_{G'}.$$

Обчислювальна складність: $O(n^k)$. Це означає, що для великих графів і великих значень k обчислення можуть потребувати значних технологічних ресурсів.

2. Ядро Вайсфейлера–Лемана (англ. *Weisfeiler-Lehman kernel*) ітеративно підраховує кількість «кольорів», що є хешованими репрезентаціями сусідства вузла:

$$c^{k+1}(v) = HASH(\{c^k(v), \{c^k(u)\}_{u \in N(v)}\}),$$

де $HASH$ – це хеш-функція, яка ставить «кольори» у відповідність вузлам. $c^k(v)$ надає інформацію про структуру сусідства вузла, яке можна охопити шляхами довжиною до k вузлів.

Обчислювальна складність: $O(|E|)$, де E – множина ребер графа.

Спосіб розв'язання задач на рівні графа

Для розв’язання задач на рівні графа використовують ядрові методи (англ. *kernel methods*) як альтернативу векторам ознак. Особливості ядер:

- ядро $K(G, G') \in \mathbb{R}$ вимірює схожість між даними;
- матриця $\boldsymbol{K} = \left(K(G, G')\right)_{G,G'}$ завжди невід’ємно визначена, тобто $\forall x \in \mathbb{C}^n : x^* \boldsymbol{K} x \geq 0$;

- існує представлення ознак $\phi(\cdot)$ таке, що $K(G, G') = \phi(G)^T \phi(G')$;
- до отриманого ядра можна застосувати метод опорних векторів (англ. *support vector machine*), щоб здійснити прогнозування.

1.3 Приклади використання графових систем для розв'язання задач бізнесу

Графова парадигма успішно застосовується у численних компаніях для розв'язання проблем, що виникають у різних бізнес-секторах [7].

1. *Виявлення шахрайства*. Фінансова компанія зі списку Fortune 500 використала графи, щоб допомогти своїм аналітикам виконати ручний аналіз складних транзакцій. За допомогою нової системи їм вдалося скоротити час на щоденну обробку 10 тис. транзакцій удвічі, заощаджуючи компанії тисячі доларів щодня.
2. *Механізми рекомендацій в реальному часі*. Платформа електронної комерції eBay запровадила графову базу даних для створення механізму рекомендацій у реальному часі свого застосунку для Google Assistant, покращуючи пошук серед великого каталогу продуктів.
3. *Графи знань*. Агентство для досліджень у галузі аеронавтики й космічних польотів NASA використало графову базу для об'єднання інформації, отриманої за 50 років дослідження космосу, виявивши закономірності та тенденції у своєму наборі даних і заощадивши понад два роки розробок і мільйон доларів від платників податків.
4. *Боротьба з «відмиванням грошей»*. Компанія грошових переказів, що переміщує майже 600 мільярдів доларів на рік, застосувала графову базу даних для виявлення штучного дроблення фінансових

операцій, що значно прискорило процеси комплаєнсу та розслідування в 150 країнах світу, в яких працює компанія.

5. *Керування основними даними*. Сервіс з короткострокової оренди житла Airbnb використав графову систему для створення свого порталу даних, який демократизував доступ до актуальної та надійної інформації в організації.
6. *Управління ланцюгами постачання*. Логістична компанія Transparency-One застосувала графову парадигму для розробки платформи моніторингу та аналізу ланцюгів постачання, забезпечення повної прозорості та управління великими обсягами взаємопов'язаних даних.
7. *Розширення можливостей управління мережевими та IT-операціями*. Телекомунікаційна компанія Telia використала графи для управління даними зі своїх маршрутизаторів, що дозволило реалізувати різноманітні сценарії, як-от підключення розумних пристроїв і підвищення ефективності обслуговування.
8. *Ідентифікація джерела даних*. Фінансова установа UBS впровадила графову систему для моделювання та управління довідковими даними про своїх клієнтів, покращивши точність інформації по всій організації.
9. *Управління ідентифікацією та доступом*. Телекомунікаційна компанія Telenor Norway замінила реляційну базу даних на графову, що значно підвищило продуктивність і масштабованість для управління контролем доступу мільйонів користувачів.
10. *Специфікація матеріалів*. Армія США використала графову систему для управління мільйонами компонентів, що значно скоротило час обробки даних і підвищило ефективність логістики.

1.4 Постановка задачі

У результаті ознайомлення з принципами використання систем графового типу в галузі бізнес-аналітики можна сформулювати постановку задачі. План виконання роботи складається з послідовності кроків, що дозволять створити комплексну систему для аналізу бізнес-даних та ухвалення рішень на основі графових структур.

1. Збір різнотипних даних із доступних наборів, а також їх попередня обробка для забезпечення узгодженості та точності інформації.
2. Побудова графової бази даних, де вузли представляють бізнес-сутності: компанії, продукти, особи, а ребра – зв’язки між ними.
3. Побудова графових нейронних мереж, їх навчання для класифікації вузлів, оцінка результатів прогнозування та порівняння різних архітектур.
4. Інтеграція системи з великою мовною моделлю для надання точних відповідей природною мовою на основі спрогнозованої інформації.
5. Створення структурованого API та інтуїтивного інтерфейсу для взаємодії із системою.

Висновки до розділу 1

Отже, графи є універсальним способом представлення об’єктів та зв’язків між ними в численних системах. Особливості графів, як-от відсутність впорядкованості об’єктів та динамічність структури, потребують спеціалізованого підходу до їх аналізу.

Основні задачі на графах: класифікація та прогнозування вузлів, зв’язків, спільнот та цілого графа. Для їх вирішення використовуються різні характеристики об’єктів, наприклад, центральність, коефіцієнт кластеризації, локальне та глобальне сусідство.

Графові системи продемонстрували високу ефективність у різноманітних галузях бізнесу: фінансах, електронній комерції, логістиці, телекомунікаціях. Вони допомагають розв'язувати задачі надання рекомендацій, управління даними та ланцюгами постачання, виявлення шахрайства тощо.

Ключовим завданням роботи є розробка графової інформаційно-аналітичної системи з використанням графової бази даних, нейронної мережі та великої мовної моделі для покращення аналізу бізнес-даних та ухвалення рішень.

РОЗДІЛ 2 ЗАСТОСУВАННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ АНАЛІЗУ ГРАФОВИХ СТРУКТУР У БІЗНЕСІ

2.1 Огляд алгоритмічних підходів для побудови векторних представлень графових бізнес-даних

Традиційне машинне навчання значною мірою спирається на отримання корисного представлення ознак даних. Графові моделі дозволяють провести *навчання представлень* (англ. *representation learning*), щоб автоматично розпізнати важливі аспекти інформації. Задачею representation learning є навчити модель встановлювати таку відповідність f між вузлами та їхніми *вкладеннями* (англ. *embeddings*) [2]:

$$f: v \to \mathbb{R}^d,$$

де v – вузол;

d – розмірність простору embeddings (англ. *latent space*), причому схожі вузли в мережі отримують близькі embeddings у latent space.

Приклад перетворення вузлів набору даних *Zachary's Karate Club* в embeddings представлений на рис. 2.1.

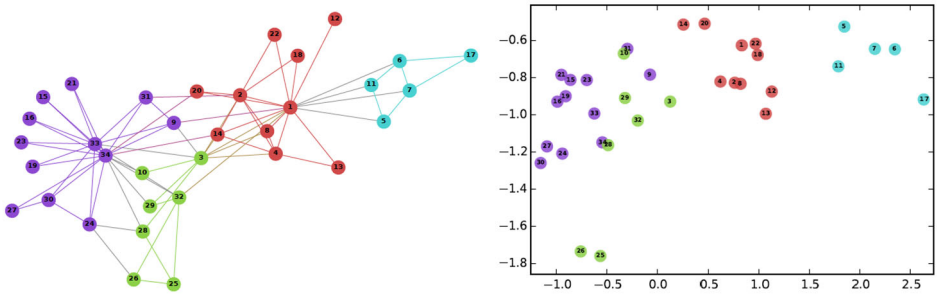

Рисунок 2.1 – Перетворення вузлів на embeddings [8]

Задача машинного навчання на графах: зробити прогнози для набору об'єктів, що мають d-розмірні вектори ознак. Тобто дано граф $G(V, E)$ і потрібно навчити модель встановлювати відповідність $f(x_i) = z_{x_i}$, бажаною властивістю якої є $\sigma(x_i, x_j) \approx z_{x_i}^T z_{x_j}$, де x – вузол, зв'язок, спільнота або цілий граф, $\sigma(x_i, x_j)$ – схожість між x_i та x_j. Для встановлення такої відповідності існує низка алгоритмів.

Алгоритм DeepWalk

З кожного вузла $v \in V$ ініціюємо випадкове блукання довжини L [8]. Випадкове блукання – це послідовність вузлів $W_v = [v_1, v_2, \dots, v_L]$, де вузол v_{i+1} – випадково обраний сусід вузла v_i. Така стратегія була обрана через інтуїтивне сприйняття схожості вузлів: якщо випадкове блукання починається у v_i та проходить v_j з великою ймовірністю, то вузли v_i та v_j є схожими. Також цей метод дозволяє розглядати не всі можливі пари вузлів, а тільки ті, які зустрічаються на одному випадковому блуканні, що підвищує ефективність алгоритму.

DeepWalk використовує модель skip-gram [9], щоб навчитися представлень вузлів. Задача полягає в оптимізації параметрів моделі так, щоб ймовірність спостерігати згенеровані випадковим блуканням послідовності вузлів була найбільшою:

$$\max_f \sum_{v_i \in V} \log P\left(\{v_{i-w}, \dots, v_{i+w}\} \middle| z_{v_i}\right),$$

де w – ширина вікна у випадковому блуканні, яка впливає на максимальну відстань взаємного впливу вузлів, а $P\left(\cdot \middle| z_{v_i}\right)$ моделюється за допомогою функції *softmax*:

$$P\left(v_j \middle| z_{v_i}\right) = \frac{\exp\left(z_{v_j}^T z_{v_i}\right)}{\sum_{u \in V} \exp\left(z_u^T z_{v_i}\right)}.$$

Тоді цільова функція, яку потрібно оптимізувати:

$$\mathcal{L} = \sum_{v_i \in V} \sum_{v_j \in N(v_i)} -\log(P(v_j|z_{v_i})),$$

де $N(v_i)$ – сусідні до v_i вузли.

Мінімізація цільової функції відбувається за допомогою *градієнтного спуску*. Обчислювальна складність: $O(|V|^2)$, де V – множина вершин графа.

Для зменшення обчислювальної складності алгоритму використовують техніку *вибірки негативних прикладів* (англ. *negative sampling*) [10]. Замість використання функції softmax, що вимагає обчислення та нормалізації відносно всіх вузлів для кожного тренувального прикладу, negative sampling використовує іншу тренувальну стратегію. Для кожного прикладу v_j, що з'являється разом з v_i на випадковому блуканні, обираються k негативних прикладів (вузли, що не з'являються у вікні v_i). Модифікована функція:

$$\log\left(\sigma\left(z_{v_j}\right)^T z_{v_i}\right) + \sum_{n=1}^{k} \mathbb{E}_{v_n \sim P_n(v)}[\log \sigma(-z_{v_n}^T z_{v_i})],$$

де $\sigma(x) = \frac{1}{1+e^{-x}}$ – *сигмоїда*;

$P_n(v)$ – розподіл, з якого отримують негативні приклади.

Для $P_n(v)$ часто обирають $freq(v)^{3/4}$, де $freq(v)$ – частота вузла v у наборі даних. Тоді нова цільова функція:

$$\mathcal{L} = \sum_{v_j \in N(v_i)} -\log P(v_j|z_{v_i}).$$

Мінімізація цільової функції відбувається за допомогою *стохастичного*

градієнтного спуску.

Алгоритм node2vec

Алгоритм node2vec розширює можливості DeepWalk за допомогою використання стратегії випадкових блукань другого порядку [11], балансуючи між пошуком у ширину (англ. *breadth-first search*) та пошуком у глибину (англ. *depth-first search*), щоб ефективно охопити і локальні, і глобальні структури графа. Генерація випадкових блукань контролюється параметром повернення p, що впливає на ймовірність одразу повернутися в попередній вузол, та параметром q, високе значення якого робить пошук подібним BFS, а низьке – DFS. Це забезпечує можливість регулювання балансу між дослідженням структур різних рівнів, що підвищує точність і корисність отриманих векторних подань.

Нехай випадкове блукання спричинило перехід з вузла t у вузол v. Тоді ймовірність P перейти у сусідній вузол x обчислюється як:

$$P(c_{i+1} = x | c_i = v, c_{i-1} = t) = \frac{\pi_{vx}}{\sum_{x' \in N(v)} \pi_{vx'}},$$

де c_i – поточний вузол випадкового блукання, π_{vx} обчислюється як:

$$\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx},$$

де w_{vx} – ваги зв’язків, а α_{pq} обчислюється як:

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p}, d_{tx} = 0 \\ 1, d_{tx} = 1, \\ \frac{1}{q}, d_{tx} = 2 \end{cases}$$

де d_{tx} – довжина найкоротшого шляху між t та x.

Приклад розрахунку значень α_{pq} зображений на рис. 2.2.

Отже, node2vec не тільки підвищує точність подання вузлів завдяки контрольованим випадковим блуканням, але й забезпечує гнучкість у налаштуванні моделі для широкого спектра завдань аналізу графів у бізнесі.

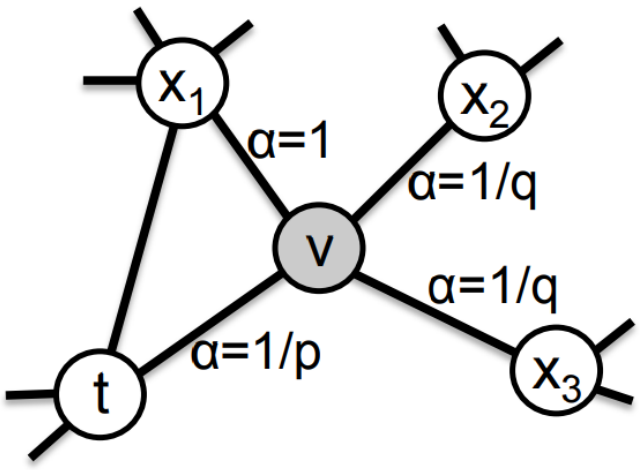

Рисунок 2.2 – Розрахунок значень α_{pq} [11]

Підходи до створення embeddings на рівні графів:

- обчислити z_G як суму embeddings окремих вузлів [12]:

$$z_G = \sum_{v \in V} z_v;$$

- увести «віртуальний вузол», що представляє весь граф, і знайти embedding цього вузла [13];
- ієрархічно кластеризувати вузли графа і знайти суму embeddings цих кластерів.

Використання embeddings вузлів:

- кластеризація/виявлення спільнот вузлів на основі z_v;
- класифікація вузлів на основі z_v;
- прогнозування зв'язків (v_i, v_j) на основі $\left(z_{v_i}, z_{v_j}\right)$;

- конкатенація: $f\left(z_{v_i}, z_{v_j}\right) = g([z_{v_i}, z_{v_j}])$;
- добуток Адамара: $f\left(z_{v_i}, z_{v_j}\right) = g(z_{v_i} \odot z_{v_j})$;
- сума: $f\left(z_{v_i}, z_{v_j}\right) = g(z_{v_i} + z_{v_j})$;
- відстань: $f\left(z_{v_i}, z_{v_j}\right) = g\left(\left\|z_{v_i} - z_{v_j}\right\|_2\right)$;

– класифікація графів на основі z_G.

Недоліки алгоритмів DeepWalk та node2vec:

– трансдуктивність: неможливо отримати embeddings вузлів, що не були присутні в тренувальній вибірці;
– погана оцінка структурної схожості віддалених елементів графа;
– необхідно використати $O(|V|)$ параметрів;
– неможливо використати ознаки на рівнях вузлів, зв'язків та графів.

Алгоритм PageRank

Для отримання значень відносної важливості вузлів у бізнес-даних можна використати алгоритм PageRank. Він був розроблений засновниками Google Ларрі Пейджем і Сергієм Бріном під час їхньої роботи над проєктом про новий вид інформаційно-пошукової системи в Стенфордському університеті. Алгоритм засновується на представленні мережевої павутини у вигляді графа, в якому вебсторінки є вузлами, а гіперпосилання – напрямленими ребрами з однієї сторінки в іншу. Головна ідея полягає в тому, що на важливі сторінки найімовірніше посилається багато інших важливих сторінок [14].

Принцип роботи PageRank:

– кожне посилання зі сторінки A на сторінку B розглядається як «голос», що віддає сторінка A за сторінку B; важливість сторінки визначається кількістю та якістю отриманих голосів;
– якість голосів, які віддає сторінка, визначаються власним PageRank, тобто посилання з важливих сторінок є більш вагомими, ніж посилання з менш авторитетних сторінок;

- PageRank моделює поведінку «випадкового мандрівника», що випадковим чином переходить за посиланнями й періодично стрибає на випадкову сторінку з усієї мережі;
- коефіцієнт демпінгу d забезпечує збіжність алгоритму.

Алгоритм підрахунку PageRank є ітеративним.

1. На початку процесу всі сторінки ініціалізуються з однаковим значенням PageRank.
2. Поки не отримали збіжність, проводять ітерації за формулою:

$$PR(u) = \frac{1-d}{N} + d \sum_{v \in B_u} \frac{PR(v)}{L(v)},$$

де $PR(u)$ – PageRank сторінки u;

N – загальна кількість сторінок;

B_u – множина сторінок, що посилаються на u;

$L(v)$ – кількість вихідних посилань на сторінці v;

d – коефіцієнт демпінгу (зазвичай рівний 0.85), що оцінює ймовірність продовження користувачем переходу за посиланнями;

$1 - d$ – ймовірність початку нового пошуку.

3. Отримані значення PageRank вказують на ймовірність знаходження «випадкового мандрівника» на відповідній сторінці у випадковий момент часу.

З наведених вище властивостей PageRank випливає, що він є розподілом стаціонарного процесу випадкових блукань, що відповідає головному власному вектору стохастичної матриці суміжності M.

Обмін повідомленнями

У бізнес-мережах часто спостерігається кореляція між характеристиками вузлів та їхньою поведінкою. Явище кореляції розглядають зі сторін гомофілії

та впливу, щоб аналізувати складні моделі взаємодії та ухвалювати стратегічні рішення у різноманітних галузях.

Явище гомофілії: індивідуальні характеристики вузлів спричиняють появу зв'язків між ними, що зображено на рис. 2.3. Наприклад, маркетингові компанії прагнуть виявляти та використовувати ефект гомофілії, щоб спрямовувати рекламу на групи клієнтів із подібними уподобаннями для кращого персоналізованого продажу товарів. Для фінансових компаній важливо виявляти мережі, що демонструють підозрілу поведінку, наприклад, гомогенні групи рахунків, залучені у шахрайських схемах.

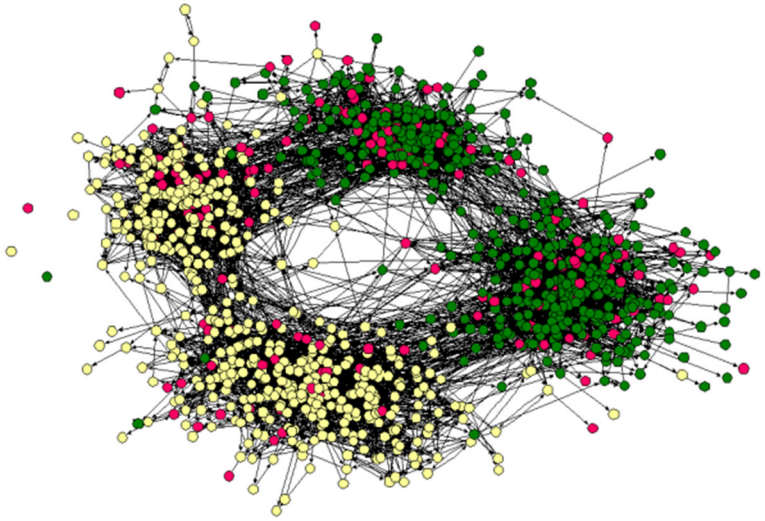

Рисунок 2.3 – Соціальна мережа американської школи ілюструє принцип гомофілії за двома характеристиками: расова ознака та дружні стосунки у середній та старшій школах [15]

Явище впливу: зв'язки між вузлами спричиняють появу спільних характеристик між ними. Ефект впливу використовують для маркетингу, спонукаючи впливових клієнтів поширювати інформацію серед своїх контактів.

Ці принципи покладено в основу підходу *колективної класифікації*: одночасно класифікують зв'язані вузли, використовуючи кореляції між ними.

Такий підхід є ймовірнісним і передбачає дотримання *марковської властивості*: мітка Y_{v_i} вузла v_i залежить від міток його сусідів N_{v_i}:

$$P\left(Y_{v_i}\right) = P(Y_{v_i}|N_{v_i}).$$

Колективна класифікація складається з трьох кроків.

1. Локальний класифікатор призначає початкові мітки на основі атрибутів вузла. На навчальній вибірці тренують звичайний класифікатор, наприклад *наївний баєсів класифікатор* або *дерево рішень*:

$$Y_{v_i} = \phi_1(f_{v_i}),$$

де f_{v_i} – вектор ознак.

Для кожного вузла створюють вектор z_{v_i}, що певним чином підсумовує початкові мітки сусідів.

2. Реляційний класифікатор призначає наступні мітки на основі попередніх міток та атрибутів сусідніх вузлів:

$$Y_{v_i} = \phi_2(f_{v_i}, z_{v_i}).$$

3. Ітеративний класифікатор застосовують, поки не отримають збіжність. На кожній ітерації обчислюють z_{v_i} і застосовують реляційний класифікатор ϕ_2.

2.2 Бізнес-орієнтовані архітектури GNN

Багатошаровий перцептрон

Графові нейронні мережі засновуються на моделі багатошарового перцептрона (англ. *multilayer perceptron*). Шари MLP поєднують лінійну трансформацію та нелінійну активаційну функцію [16, 17]:

$$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)},$$
$$a^{(l)} = \sigma^{(l)}(z^{(l)}),$$

де $a^{(l)}$ – вихід шару l;

$W^{(l)}$ – матриця ваг між шарами $(l-1)$ та l;

$b^{(l)}$ – зміщення (англ. *bias*) шару l;

$\sigma^{(l)}$ – функція активації шару l.

Окрім *вхідного* та *вихідного шарів*, MLP містить один або більше *прихованих шарів*, що зумовлює використання алгоритму *зворотного поширення помилки* [18] (англ. *backpropagation*) для обчислення градієнтів. Нехай маємо MLP з чотирма шарами: вхідний $x^{(0)}$, два приховані, та вихідний, причому кожний шар містить по два нейрони. Компоненти такого MLP зображено на рис. 2.4, причому нейрони прихованого шару розбито на окремі складові $z^{(l)}$ (після лінійного перетворення) і $a^{(l)}$ (після застосування активаційної функції).

Задача машинного навчання на графах формулюється як класична задача оптимізації:

$$\min_{\theta} \mathcal{L}(y, \hat{y}),$$

де θ – множина параметрів, які прагнуть оптимізувати;

y – цільові значення;

$\hat{y}$ – передбачені значення;

$\mathcal{L}$ – функція втрат, наприклад $\mathcal{L}(y, \hat{y}) = \|y - \hat{y}\|_2$ – норма $L2$.

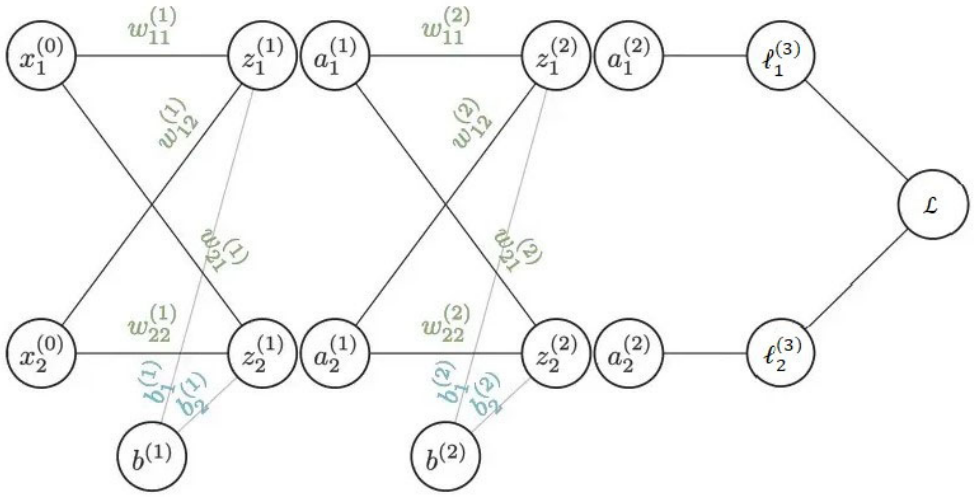

Рисунок 2.4 – Багатошаровий перцептрон[1]

Тоді щоб знайти, як змінюється $\mathcal{L}$ зі зміною параметрів мережі, скористаємося *ланцюговим правилом*:

$$\frac{\partial \mathcal{L}}{\partial W^{(2)}} = \frac{\partial \mathcal{L}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial W^{(2)}}.$$

Аналогічно знаходять градієнти відносно $b^{(l)}$ та відносно параметрів попередніх шарів. За допомогою отриманих градієнтів проводять градієнтний спуск для оптимізації параметрів мережі.

[1] Джерело зображення: https://towardsdatascience.com/understanding-backpropagation-abcc509ca9d0

Графова нейронна мережа

Окрім шарів MLP, графові нейронні мережі (англ. *graph neural networks*) мають особливі шари, що виконують кроки *обміну повідомленнями* (англ. *message passing*) та *оновлення вузлів* [19].

1. Обчислення повідомлення: вузол v використовує власні ознаки (і, можливо, ознаки суміжних ребер), щоб скласти своє повідомлення застосуванням функції MSG (наприклад, $MSG(x) = W^{(l)}(x)$):

$$m_v^{(l)} = MSG^{(l)}\left(h_v^{(l-1)}\right),$$

де $h_v^{(l-1)}$ – власний embedding вузла з попереднього шару $(l-1)$.

2. Агрегація повідомлень: вузол збирає та комбінує повідомлення сусідів шляхом застосування функції AGG ($AGG(x) \in \{SUM(x), MEAN(x), MAX(x), \dots\}$). Обрана функція повинна бути інваріантною до порядку вузлів:

$$a_v^{(l)} = AGG^{(l)}(\left\{m_u^{(l)} : u \in N(v)\right\}, m_v^{(l)}).$$

3. Оновлення вузла: вузол певним чином перетворює агреговані повідомлення, наприклад, використовуючи нелінійну активаційну функцію, і оновлює власний embedding:

$$h_v^{(l)} = UPD^{(l)}(h_v^{(l-1)}, a_v^{(l)}).$$

Цей процес у шарі GNN зображено на рис. 2.5.

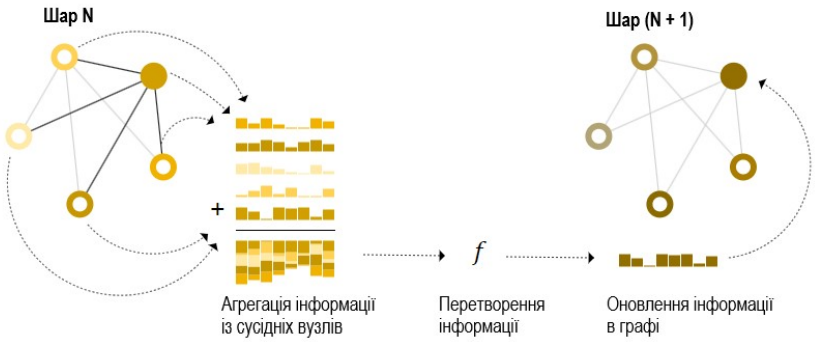

Рисунок 2.5 – Шар GNN [20]

Обираючи різні функції MSG, AGG та UPD, можна отримати численні модифікації стандартних GNN.

Графова згорткова мережа

Графові згорткові мережі [21] (англ. *graph convolutional networks*, GCN) розширюють концепцію згорткових нейронних мереж [22] (англ. *convolutional neural networks*, CNN) на графи. У той час як CNN працюють з сітками пікселів чітко заданого розміру та структури, GCN здатні обробляти довільні зв'язки між вузлами графу, який може змінюватися. Структура GCN зображена на рис. 2.6.

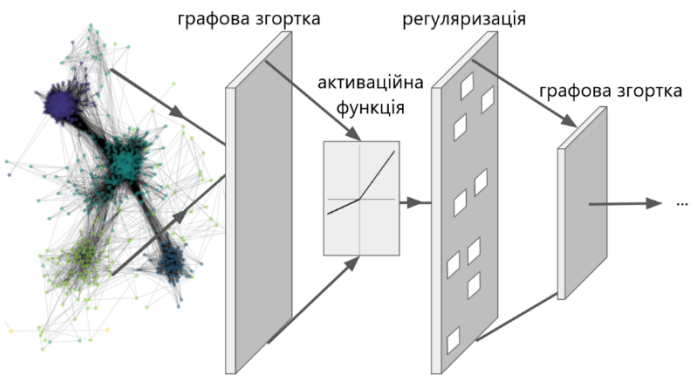

Рисунок 2.6 – Структура графової згорткової мережі[2]

[2] Джерело зображення: https://www.experoinc.com/insights/blog/node-classification-by-graph-convolutional-network

Шари GCN виконують таке перетворення:

$$h_v^{(l)} = \sigma(W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l-1)}}{|N(v)|}).$$

Графова мережа уваги

Графова мережа уваги (англ. *graph attention network*, GAT) адаптує принцип *уваги* на графи.

Шари GAT виконують таке перетворення [23]:

$$h_v^{(l)} = \sigma(W^{(l)} \sum_{u \in N(v)} \alpha_{vu} W^{(l)} h_u^{(l-1)}),$$

де α_{vu} – вага, що налаштовується в ході тренування моделі.

Механізм уваги складається з двох кроків.

1. Функція a обчислює коефіцієнти уваги e_{vu} пар вузлів v, u на основі їхніх повідомлень:

$$e_{vu} = a(W^{(l)} h_u^{(l-1)}, W^{(l)} h_v^{(l-1)}),$$

де e_{vu} вказує на важливість повідомлення вузла u для вузла v.

2. Коефіцієнти e_{vu} нормалізують за допомогою функції softmax і отримують ваги α_{vu}:

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}.$$

Такий процес навчання механізму уваги може бути нестабільним. Для підвищення надійності алгоритму використовують *багатоголову увагу* (англ. *multi-head attention*).

1. Створюють k *голів уваги* з власними параметрами:

$$h_v^{(l)\{1\}} = \sigma(\sum_{u \in N(v)} \alpha_{vu}^{\{1\}} W^{(l)} h_u^{(l-1)}),$$
$$h_v^{(l)\{2\}} = \sigma(\sum_{u \in N(v)} \alpha_{vu}^{\{2\}} W^{(l)} h_u^{(l-1)}),$$
$$\dots$$
$$h_v^{(l)\{k\}} = \sigma(\sum_{u \in N(v)} \alpha_{vu}^{\{k\}} W^{(l)} h_u^{(l-1)}).$$

2. Результати роботи голів уваги агрегують функцією AGG:

$$h_v^{(l)} = AGG(h_v^{(l)\{1\}}, h_v^{(l)\{2\}}, \dots, h_v^{(l)\{k\}}).$$

Приклад обчислених коефіцієнтів уваги на наборі даних *Cora* зображено на рис. 2.7. Бізнес-застосування механізму уваги: у системах рекомендацій він допомагає виділяти найважливіші взаємодії та атрибути користувачів і товарів; для моніторингу відгуків клієнтів він дозволяє системно фокусуватися на ключових фрагментах інформації; у фінансових додатках увага допомагає зосередитися на індикаторах, пов’язаних з аномаліями транзакцій.

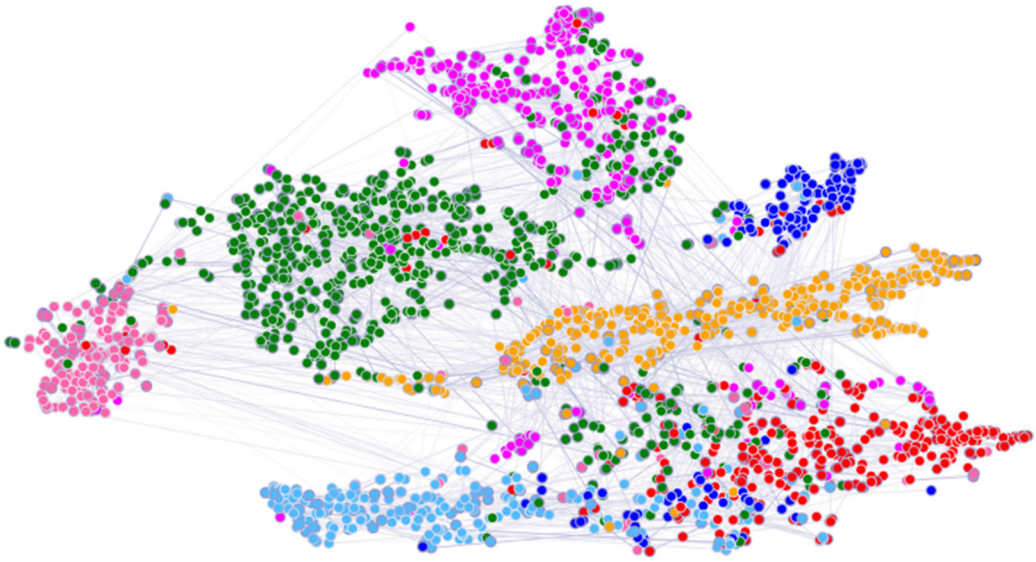

Рисунок 2.7 – t-SNE візуалізація обчислених за допомогою GAT ознак набору даних *Cora*. Кольори позначають класи вузлів, а товщина ребер – агреговані нормалізовані коефіцієнти уваги, обчислені за допомогою восьми голів уваги [23]

Переваги механізму уваги:

- використання окремих значень важливості α_{vu} сусідніх вузлів;
- паралелізація обчислення коефіцієнтів уваги на всіх ребрах графа та паралелізація агрегації коефіцієнтів уваги на всіх вузлах графа, що дозволяє бізнес-системам працювати в режимі реального часу;
- кількість тренувальних параметрів не залежить від розміру графа, тобто з'являється можливість аналізувати великі обсяги бізнес-даних;
- обчислювальна складність відповідних операцій над розрідженими матрицями обмежується $O(V + E)$;
- індуктивність: функція a не залежить від структури графа і може бути застосована на інших графах після тренування, тобто розроблена компанією система може працювати з даними різної структури та продовжувати розв'язувати задачі з високою точністю.

Модифікації шару GNN

1. *Пакетна нормалізація* [24] (англ. *batch normalization*) дозволяє стабілізувати процес навчання шляхом зведення пакетів вхідних даних до нульового середнього $\mu_{i,j}$ та одиничної дисперсії $\sigma_{i,j}^2$. Нехай маємо пакет embeddings вузлів розміром N. Тоді вихідні embeddings $Y_{i,j}$ знаходять у такий спосіб:

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} X_{i,j},$$
$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(X_{i,j} - \mu_j),$$
$$\hat{X}_{i,j} = \frac{X_{i,j}-\mu_j}{\sqrt{\sigma_j^2+\epsilon}},$$
$$Y_{i,j} = \gamma_j \hat{X}_{i,j} + \beta_j,$$

де γ_j, β_j – параметри, що зберігають виразність мережі [25].

2. *Дропаут* [26] (англ. *dropout*) дозволяє зменшити *перенавчання*, використовуючи випадкову підмережу (шляхом вимкнення нейронів з ймовірністю p) замість головної мережі під час тренування. У ході тестування використовують повну мережу.
3. Зміна функції активації, що застосовується до embedding, також може покращити процес тренування. Серед поширених функцій: $ReLU(x_i) = \max(x_i, 0)$, $PReLU(x_i) = \max(x_i, 0) + a_i \min(x_i, 0)$ та сигмоїда.

Багатошарова GNN

Використовуючи k шарів GNN, кожний вузол може агрегувати інформацію з сусідства радіуса k, що дозволяє бізнес-системам використовувати більш точну інформацію для підтримки ухвалення рішень. Але зі збільшенням кількості шарів з’являється проблема *надмірного*

згладжування (англ. *over-smoothing*), зображена на рис. 2.8: embeddings вузлів сходяться до одного значення, що унеможливлює спроби розрізнити вузли.

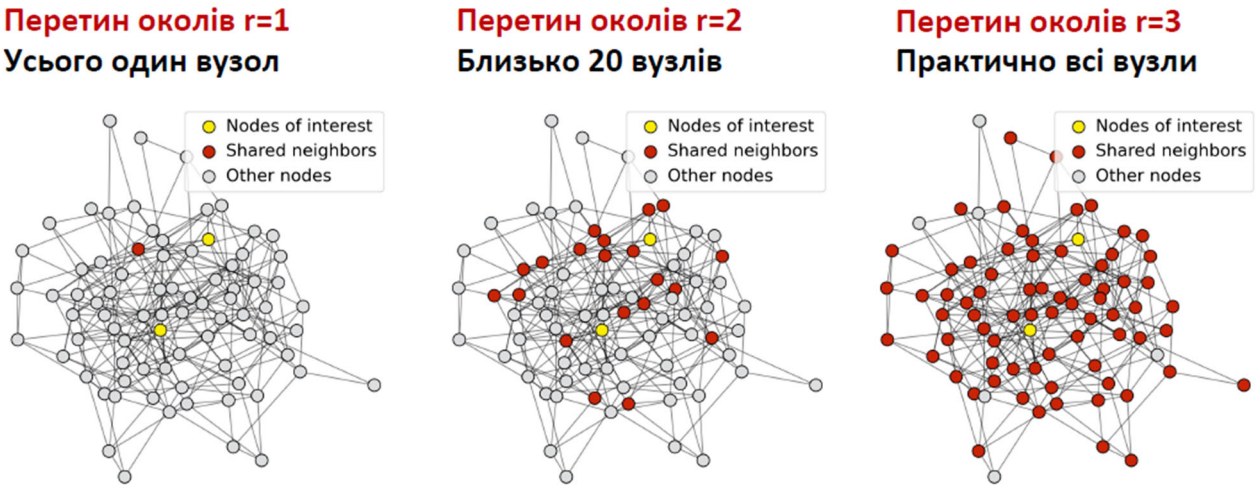

Рисунок 2.8 – Проблема надмірного згладжування з'являється зі збільшенням кількості шарів GNN [27]

Методи запобігання надмірному згладжуванню:

– аналіз необхідного розміру рецептивного поля R, наприклад, за допомогою діаметра графа D_G і встановлення кількості шарів $L \gtrsim R$;
– збільшення виразності мережі за допомогою заміни лінійної трансформації MSG та AGG на окремий MLP;
– додавання шарів попередньої обробки для покращення перетворення ознак вузлів у embeddings та шарів подальшої обробки для вдосконалення інтерпретації embeddings;
– додавання *пропускових з'єднань* (англ. *skip connections*) на рівні шару або на рівні мережі, щоб опрацювати embeddings різними шляхами всередині мережі. Наприклад, GCN з пропусковими з'єднаннями може виконувати таке перетворення:

$$h_v^{(l)} = \sigma(W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l-1)}}{|N(v)|} + h_v^{(l-1)}).$$

Аугментація графа

У багатьох випадках вхідний граф не є оптимальним для обчислення embeddings [27]. Можливі проблеми та їхні рішення представлені у табл. 2.1.

Таблиця 2.1 – Аугментація графа (англ. *graph augmentation*) допомагає подолати поширені проблеми, що виникають у вхідному графі

Проблема	*Рішення*
Недостатня кількість ознак спричиняє *недонавчання* (англ. *underfitting*)	Додати ознаки, які складно вивчити за допомогою GNN, наприклад, степінь вузла, коефіцієнт кластеризації, PageRank, центральність
Розрідженість графа робить обмін повідомленнями неефективним	Додати віртуальні вузли або зв'язки, що зменшують відстань між вузлами графа
Щільність графа робить обмін повідомленнями затратним	Використовувати лише частину сусідніх вузлів для обміну повідомленнями
Великий розмір графа унеможливлює його повне розміщення в GPU	*Вибірка сусідів* [28] (англ. *neighbor sampling*), Cluster-GCN [29], SGC [30]

2.3 Методики навчання GNN для задач прогнозування в бізнесі

Для успішного використання GNN у бізнес-застосунках необхідно навчити модель на відповідних даних. Процес навчання GNN має свої особливості порівняно з традиційними нейронними мережами, що зумовлено специфічною природою графових даних. Зокрема, потрібно враховувати залежності між вузлами та зв'язками графа, а також коректно розділяти дані на тренувальний, валідаційний та тестовий набори, уникаючи витоку даних через наявні зв'язки. Крім того, слід обирати відповідні методи прогнозування міток та функції втрат залежно від типу задачі.

Процес навчання графової нейронної мережі зображено на рис. 2.9.

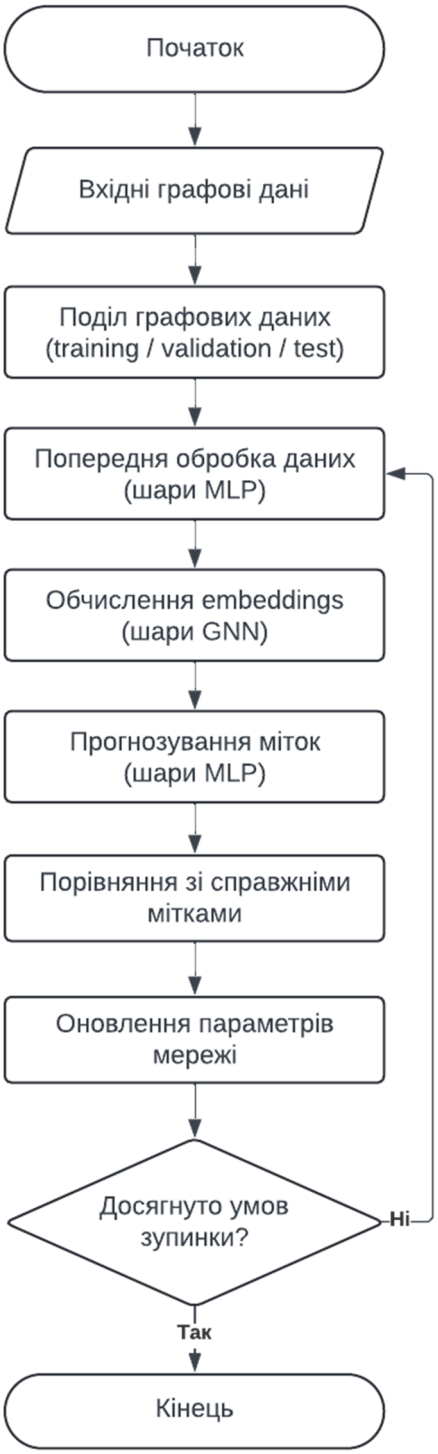

Рисунок 2.9 – Процес навчання графової нейронної мережі

Поділ графових даних

У машинному навчанні розповсюдженою практикою є ділення початкового набору даних на *тренувальний*, *валідаційний* та *тестувальний* набори. Однак використання цієї стратегії на графах часто спричиняє *витік*

даних (англ. *data leakage*) через притаманні графам залежності між елементами [27]. Це зумовлює застосування особливих модифікацій до поділу набору даних, що зображено в табл. 2.2.

Таблиця 2.2 – Особливості поділу набору даних на графах G

	Тренування (скор. $train$)	*Валідація* (скор. $valid$)	*Тестування* (скор. $test$)
Рівень вузлів			
Трансдуктивність	$G \to V_{train}$	$G \to V_{valid}$	$G \to V_{test}$
Індуктивність	$G_{train} \to V_{train}$	$G_{valid} \to V_{valid}$	$G_{test} \to V_{test}$
Рівень зв'язків			
Трансдуктивність	$E_{train_{msg}} \to E_{train_{sup}}$	$E_{train_{msg}} \cup E_{train_{sup}} \to E_{valid}$	$E_{train_{msg}} \cup E_{train_{sup}} \cup E_{valid} \to E_{test}$
Індуктивність	$E_{train_{msg}} \to E_{train_{sup}}$	$E_{valid_{msg}} \to E_{valid_{sup}}$	$E_{test_{msg}} \to E_{test_{sup}}$
Рівень графа			
Трансдуктивність	N/A	N/A	N/A
Індуктивність	$G_{train} \to G_{train}$	$G_{valid} \to G_{valid}$	$G_{test} \to G_{test}$

Причому:

– трансдуктивність: дані складаються з єдиного графа G

$$G = (V, E),$$
$$G_{train} = (V_{train}, E_{train}),$$
$$G_{valid} = (V_{valid}, E_{valid}),$$
$$G_{test} = (V_{test}, E_{test}),$$
$$V = V_{train} \sqcup V_{valid} \sqcup V_{test},$$
$$E \supset E_{train} \sqcup E_{valid} \sqcup E_{test};$$

– індуктивність: дані складаються з багатьох графів $\mathcal{G} = \mathcal{G}_{train} \sqcup \mathcal{G}_{valid} \sqcup \mathcal{G}_{test} = \{G_1, G_2, \dots\}$;

– $X \to Y$: X – це дані, на основі яких знаходять embeddings, Y – це дані, що прогнозують;
– залежності між зв’язками зумовлюють окремий поділ на $E_{i_{msg}}$ – зв’язки повідомлень і $E_{i_{sup}}$ – зв’язки контролю.

Прогнозування міток

Процес знаходження міток залежить від рівня задачі [27]:

– на рівні вузлів використовують embedding вузла v:

$$\widehat{y_v} = f(h_v^{(l)}),$$

де $f(v)$ – певна лінійна трансформація $Linear(v)$;

– на рівні зв'язків використовують embeddings вузлів u та v:

$$\widehat{y_{u,v}} = f(h_u^{(l)}, h_v^{(l)}),$$

де $f(u, v) \in \{Linear(Concat(u, v)), u^T v, \dots\}$;

– на рівні графа використовують embeddings усіх вузлів графа G:

$$\widehat{y_G} = f(\{h_v^{(l)}, \forall v \in G\}),$$

де $f(G) \in \{Mean(G), Max(G), Sum(G), DiffPool(G)[31], \dots\}$.

Вигляд міток залежить від типу задачі [27]:

– кероване навчання: модель має доступ до міток елементів графа;
  - класифікація: мітки $y^{(i)}$ мають дискретні значення;
  - регресія: мітки $y^{(i)}$ мають неперервні значення;

- некероване навчання: модель використовує лише структурні особливості графа.

Порівняння зі справжніми мітками

Для оптимізації параметрів мережі та оцінки її якості під час навчання використовують *функцію втрат* [27]:

- для класифікації функцією втрат є *перехресна ентропія* (англ. *cross-entropy, CE*):

$$CE(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log \hat{y}_i;$$

- для регресії функцією втрат є *середньоквадратична похибка* (англ. *mean squared error, MSE*):

$$MSE(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2.$$

Для оцінки якості моделі під час валідації та тестування використовують *метрики* [27]:

- для класифікації метриками є *точність* (англ. *accuracy*), влучність (англ. *precision*), *повнота* (англ. *recall*), *F_1-score*, …

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN},$$
$$Precision = \frac{TP}{TP+FP},$$
$$Recall = \frac{TP}{TP+FN},$$
$$F_1 = 2\frac{Precision \cdot Recall}{Precision+Recall},$$

де результат TP – істинно позитивний;

TN – істинно негативний;

FP – хибно позитивний (помилка I роду);

FN – хибно негативний (помилка II роду);

– для регресії метриками є *корінь із середньоквадратичної похибки* (англ. *root mean squared error, RMSE*), *середня абсолютна похибка* (англ. *mean absolute error, MAE*), …

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2},$$

$$MAE(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|.$$

Оновлення параметрів мережі

У класичних нейронних мережах оновлення параметрів виконує *оптимізатор*, який для цього використовує отримані за допомогою backpropagation градієнти. Наприклад, за оптимізатор можна обрати *градієнтний спуск*.

1. Випадковим чином ініціалізують θ.
2. Поки не отримують збіжність, роблять кроки у напрямку, протилежному градієнту $\nabla\mathcal{L}$: $\theta \leftarrow \theta - \eta\nabla\mathcal{L}(\theta)$, де η – коефіцієнт швидкості навчання.

Недоліком градієнтного спуску є висока обчислювальна складність, що спричиняє повільну збіжність на великих наборах даних [27].

Для прискорення збіжності можна використати *стохастичний градієнтний спуск*, що отримує градієнт лише від одного навчального екземпляра, але через це є менш стабільним. Для отримання компромісу між швидкістю та стабільністю використовують *мініпакетний градієнтний спуск*, що одержує значення градієнта від підмножини всіх екземплярів.

Проте якщо навчальні приклади – це вузли великого графа, то мініпакетний градієнтний спуск не можна застосувати в оригінальному вигляді,

адже обрані вузли з великою ймовірністю будуть знаходитися на великій відстані та не зможуть ефективно виконувати обмін повідомленнями.

Алгоритм GraphSAGE [28] натомість використовує *вибірку сусідів*: разом з кожним вузлом v обираються випадкові H_k вузлів з кожного рівня $k = 1,2,\dots,K$ сусідства вузла v. Особливості GraphSAGE:

- обчислювальна складність: $O(H^K)$;
- зменшення параметра H призводить до зменшення обчислювальної складності алгоритму і збільшення *дисперсії*;
- стратегія вибору сусідніх вузлів може бути покращена заміною на *випадкове блукання з перезапусками* (як в алгоритмі PageRank): обираються найкращі H_k вузлів за оцінкою R_i.

Шари GraphSAGE виконують таке перетворення:

$$h_v^{(l)} = \sigma(W^{(l)} \cdot CONCAT\left(h_v^{(l-1)}, AGG\left(\left\{h_u^{(l-1)} : u \in N(v)\right\}\right)\right)).$$

2.4 Імплементація GNN на графових наборах бізнес-даних

Гетерогенні графи

Розглянемо застосування GNN до гетерогенних графів, що є типовими для бізнес-даних. Гетерогенні графи містять різні типи вузлів та зв'язків, що відображають складну структуру сутностей у бізнес-середовищі. Одним із прикладів гетерогенного графа може бути мережа взаємодії між компаніями, клієнтами, продуктами та постачальниками. У такому графі можуть бути вузли різних типів: «компанія», «клієнт», «продукт», «постачальник», а зв'язки між ними можуть позначати різні типи відносин, такі як «постачає», «купує», «виробляє» тощо. Гетерогенні графи позначаються $G = (V, E, \phi, \psi)$, де V – вузли, E – зв'язки, $\phi: V \to \mathcal{T}_V$ – функція, що ставить у відповідність кожному

вузлу з V тип з $\mathcal{T}_V$, $\psi: E \to \mathcal{R}_E$ – функція, що ставить у відповідність кожному зв'язку з E тип відношення з $\mathcal{R}_E$. Приклад такого графа зображено на рис. 2.10.

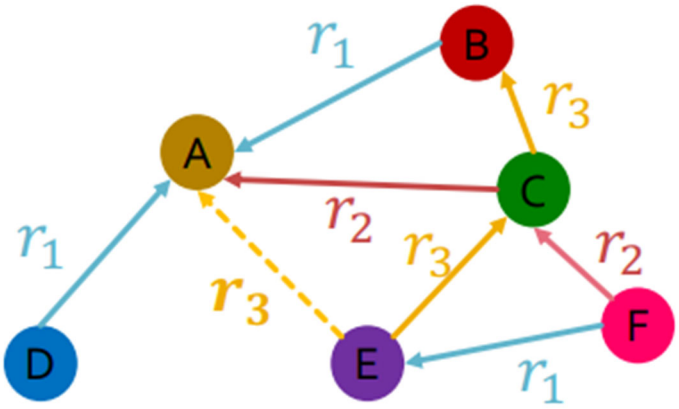

Рисунок 2.10 – Гетерогенний граф [27]

Для адаптації GCN до гетерогенних графів можна використати RGCN [32]:

$$h_v^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}} \sum_{u\in N_v^r} \frac{1}{c_{v,r}} W_r^{(l)} h_u^{(l)} + W_0^{(l)} h_v^{(l)}\right),$$

де $W_r^{(l)}$ – ваги, що відрізняються не тільки за шаром, а і за типом відношення; $c_{v,r} = |N_v^r|$.

Поява окремих матриць ваг для кожного типу відношення спричиняє значне зростання обчислювальної складності, тому для $W_r^{(l)}$ зазвичай використовують блочні діагональні матриці або дозволяють ділитися параметрами з іншими $W_r^{(l)}$.

Прогнозування зв'язків для RGCN також зазнає певних змін [27]. Нехай маємо задачу прогнозування появи зв'язку контролю (E, r_3, A) з рис. 2.10.

1. Для кожного r_i відбувається поділ на окремі тренувальні, валідаційні та тестові множини, щоб зберегти пропорційність r_i.

2. Використовують RGCN для оцінки зв'язку (E, r_3, A).
3. Створюють негативний приклад (E, r_3, B), що не може бути зв'язком повідомлень.
4. Використовують RGCN для оцінки зв'язку (E, r_3, B).
5. У ролі функції втрат використовують перехресну ентропію:

$$\mathcal{L} = -\log \sigma\left(f_{r_3}(h_E, h_A)\right) - \log(1 - \sigma\left(f_{r_3}(h_E, h_B)\right)).$$

Графи знань

Графи знань (англ. *knowledge graphs*) є окремим випадком гетерогенних графів, що представляють об'єкти реального світу та відношення між ними у графовій формі. Приклади графів знань:

- Google Knowledge Graph [33] використовується в пошуковій системі Google для надання релевантної інформації користувачам, містить мільярди об'єктів і сотні мільярдів зв'язків між ними;
- Facebook Graph API [34] надає розробникам доступ до соціального графа Facebook, що містить користувачів, фотографії, події, сторінки тощо;
- YAGO [35] комбінує інформацію з Wikidata та schema.org, отримуючи мільярди фактів і зручну онтологію.

У бізнес-сфері графи знань мають застосування, наприклад, в аналізі конкурентного середовища. Графи знань можуть представляти компанії, їхні продукти, стратегії, фінансові показники, а також взаємозв'язки між ними як конкурентами, партнерами, постачальниками тощо.

Графи знань зазвичай містять велику кількість вузлів, але матриця зв'язків між ними є досить розрідженою, тому постає задача доповнення графа знань (англ. *knowledge graph completion*).

Зв'язки у графі знань представлені у вигляді трійок (h, r, t), де h - «голова», r – відношення, t – «хвіст». Задача: передбачити t, маючи (h, r). Для

цього використовують інтуїтивний принцип: embedding $(\boldsymbol{h}, \boldsymbol{r})$ повинен бути близьким до embedding $\boldsymbol{t}$. Різні способи визначення самих embeddings і їхньої близькості зумовлюють появу різних методів доповнення.

Деякі з поширених методів доповнення графа знань представлені в табл. 2.3.

Таблиця 2.3 – Методи доповнення графа знань [27]

Модель	*Embedding*	*Оцінка*
TransE [36]	$\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{R}^k$	$-\|\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}\|$
TransR [37]	$\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{R}^k$ $\boldsymbol{W_r} \in \mathbb{R}^k$	$-\|\boldsymbol{W_r h} + \boldsymbol{r} - \boldsymbol{W_r t}\|$
DistMult [38]	$\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{R}^k$	$< \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} >$
ComplEx [39]	$\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{C}^k$	$Re(< \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} >)$

Властивості методів доповнення графа знань:

- симетричність: $\forall h, t: r(h, t) \Rightarrow r(t, h)$;
- антисиметричність: $\forall h, t: r(h, t) \Rightarrow \neg r(t, h)$;
- оберненість: $\forall h, t: r_2(h, t) \Rightarrow r_1(t, h)$;
- транзитивність: $\forall x, y, z: r_1(x, y) \wedge r_2(y, z) \Rightarrow r_3(x, z)$;
- один-до-багатьох (1:N): $\exists h, t_1, t_2, \ldots t_n: r(h, t_1) \wedge r(h, t_2) \wedge \ldots \wedge \wedge r(h, t_n) \wedge (\forall i \neq j: t_i \neq t_j)$.

Властивості розглянутих вище методів представлені в табл. 2.4.

Таблиця 2.4 – Властивості методів доповнення графа знань [27]

Модель	*Сим.*	*Антисим.*	*Оберн.*	*Транз.*	*1:N*
TransE	×	✓	✓	✓	×
TransR	✓	✓	✓	×	✓
DistMult	✓	×	×	×	✓
ComplEx	✓	✓	✓	×	✓

Виконання запитів на графах знань

Одноступеневий запит: знайти місто, в якому знаходиться штаб-квартира компанії Acme Corp (рис. 2.11):

$$q = (v: AcmeCorp, r: HasHeadquartersIn).$$

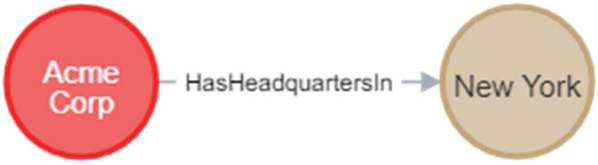

Рисунок 2.11 – Одноступеневий запит

Шляховий запит: знайти постачальників компонентів продукту (рис. 2.12):

$$q = (v: Gadget, r: HasComponent, r: SuppliedBy).$$

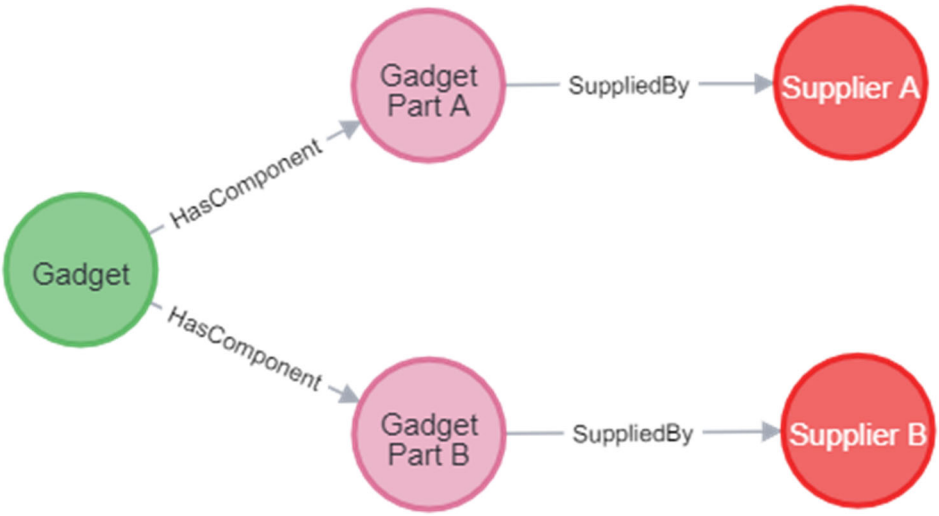

Рисунок 2.12 – Шляховий запит

Кон'юнктивний запит: знайти особу, що є членом ради директорів Acme Corp та інвестором Tech Corp (рис. 2.13):

$$q = (v: AcmeCorp, r: HasBoardMember) \wedge$$
$$\wedge (v: TechCorp, r: HasInvestor).$$

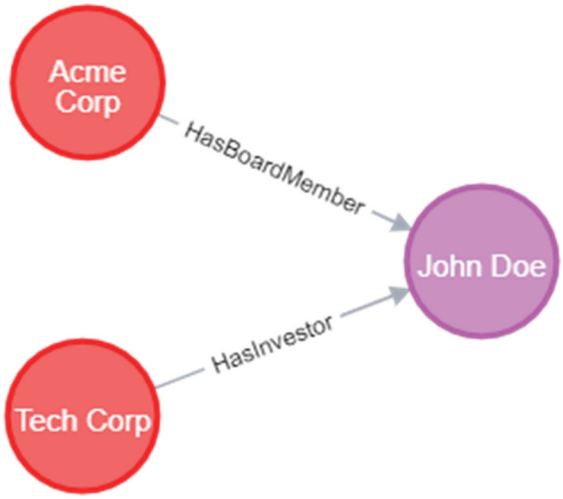

Рисунок 2.13 – Кон'юнктивний запит

Прогностичні запити

Коли граф містить усі дані, можна отримати відповідь на питання за допомогою використання мови запитів відповідної графової бази даних. Однак якщо база даних не містить потрібних вузлів і зв'язків, постає задача відповіді на *прогностичний запит* (англ. *predictive query*). Раніше розглянуті методи доповнення графа створюють щільний ймовірнісний граф і призводять до високої обчислювальної складності $O(d_{max}^{L})$, де d_{max} – максимальний степінь вузла, L – довжина шляхового запиту.

Ефективним методом надання відповіді на прогностичні шляхові запити є узагальнення алгоритму TransE на багатоступеневе прогнозування [40]: знаходять embedding запиту $\boldsymbol{q} = \boldsymbol{h} + \boldsymbol{r}$ і намагаються зробити його якомога ближчим до embedding відповіді $\boldsymbol{t}$.

Нехай у базі даних зберігаються вузли енергетичних компаній, електростанцій та індустрій, але відсутня інформація про зв'язки між вузлами. Процес знаходження індустрій, що залежать від енергетичної компанії Energy Corp, представлений на рис. 2.14.

1. Обчислюють embedding вузла Energy Corp $\mathbf{h}$ та зв'язків HasPlant $\mathbf{r_1}$ і SuppliesTo $\mathbf{r_2}$.
2. Знаходять найкращі збіги електростанцій $\boldsymbol{t_1}$ в околі $\mathbf{h} + \mathbf{r_1}$.
3. Знаходять найкращі збіги індустрій $\boldsymbol{t_2}$ в околі $\mathbf{t_1} + \mathbf{r_2}$.

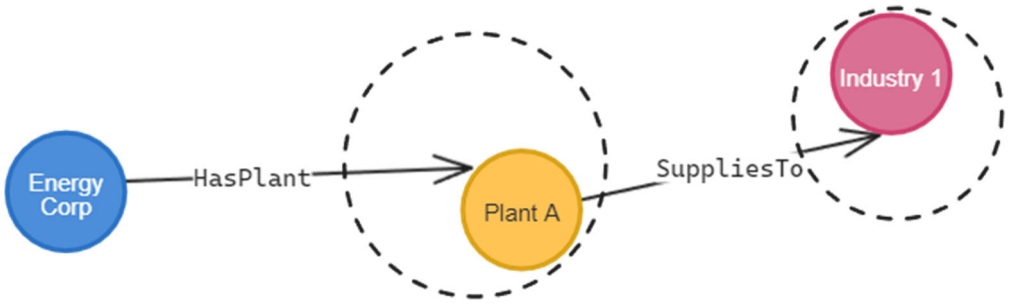

Рисунок 2.14 – Знаходження індустрій, що залежать від енергетичної компанії Energy Corp

Для знаходження відповіді на прогностичні кон'юнктивні запити використовують алгоритм Query2Box [41]. Основна ідея полягає у застосуванні гіперпрямокутника (англ. *box*) $\boldsymbol{q}$, що характеризується центром $Center(\boldsymbol{q})$, та відстанню від кута до центра – $Offset(\boldsymbol{q})$ (рис. 2.15).

1. Кожний вузол розглядається як box нульового об'єму.
2. Кожний зв'язок перетворює вхідний box на вихідний за допомогою оператора проєкції $\mathcal{P}: box \times r \to box$.
3. Кон'юнкція запитів знаходиться як перетин $\mathcal{J}: box \times \ldots \times box \to box$, причому:

$$Center(\boldsymbol{q_{inter}}) = \sum_i \boldsymbol{w_i} \odot Center(\boldsymbol{q_i}),$$

$$\boldsymbol{w_i} = \frac{exp(f_{cen}(Center(\boldsymbol{q_i})))}{\sum_i exp(f_{cen}(Center(\boldsymbol{q_i})))},$$

де $\boldsymbol{q_{inter}}$ – embedding кон'юнкції;

f – оператор перетину.

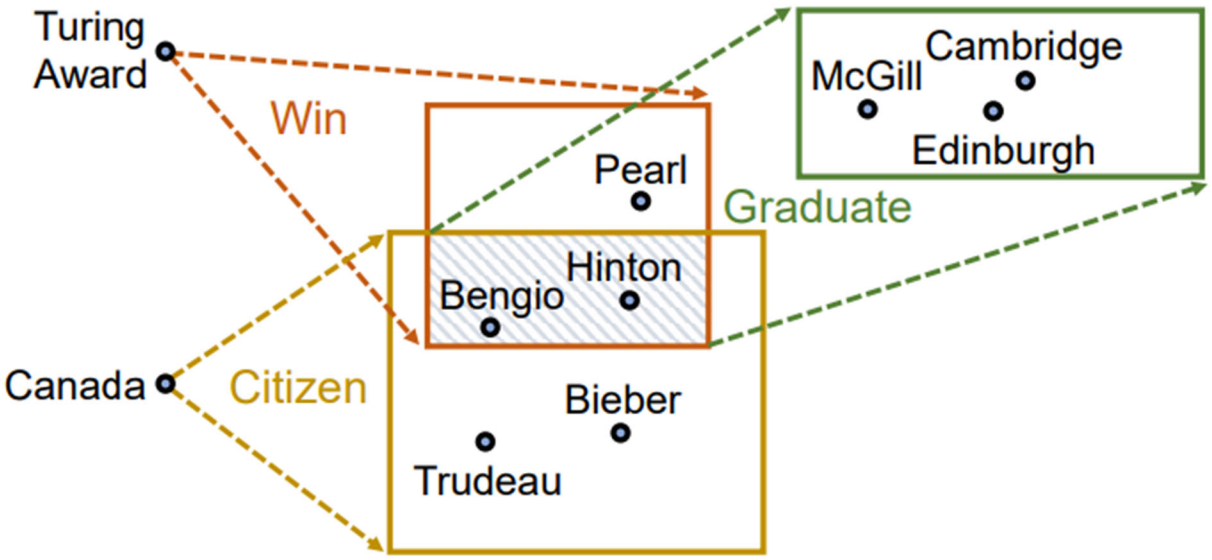

Рисунок 2.15 – Знаходження *alma mater* громадян Канади, що отримали премію Тюрінга [41]

Висновки до розділу 2

Отже, машинне навчання на графах має на меті надання прогнозів за допомогою векторних представлень (embeddings) вузлів, зв'язків та цілих графів; ці представлення відображають структурні властивості та схожість елементів.

Найпоширеніші алгоритми для отримання embeddings включають DeepWalk (з використанням випадкових блукань і skip-gram), node2vec (з використанням зміщених випадкових блукань) та алгоритми, засновані на передачі повідомлень між вершинами, як-от графові згорткові мережі (GCN) та графові мережі уваги (GAT).

Для збільшення ефективності процесу навчання і запобігання надмірному згладжуванню використовують вибірку сусідів, пакетну нормалізацію, dropout,

пропускові з’єднання і доповнення графа. Індуктивні можливості дозволяють застосовувати навчені графові нейронні мережі до нових графів, тоді як трансдуктивні методи обмежуються початковим графом.

Методи DistMult і ComplEx допомагають знайти embeddings сутностей і зв’язків у графі знань, щоб відповісти на шляхові та кон’юнктивні запити, а модифікований метод TransE і Query2Box застосовують для прогностичних запитів.

РОЗДІЛ 3 РОЗРОБКА ГРАФОВОЇ СИСТЕМИ ПІДТРИМКИ БІЗНЕС-РІШЕНЬ

Створення графової системи підтримки рішень у бізнесі та імплементація описаних раніше алгоритмів потребують сучасних та ефективних інструментів реалізації бази даних, графової нейронної мережі, великої мовної моделі та серверної частини застосунку.

Для розробки системи була обрана мова програмування Python. Ця мова має простий та читабельний синтаксис, що прискорює процес розробки та зменшує ймовірність помилок. Вона підтримує процедурне, об’єктно-орієнтоване та функціональне програмування, тобто є гнучким інструментом написання логіки програми. Python має велику кількість бібліотек та фреймворків для широкого спектра задач від аналізу даних до машинного навчання. Також ця мова легко інтегрується з іншими технологіями, забезпечуючи ефективну взаємодію з базами даних та вебсервісами. Реалізація вебкомпонентів відбувається за допомогою мікрофреймворку Flask, який надає основні інструменти для веброзробки й дозволяє розширити функціональність шляхом інтеграції додаткових плагінів за потреби.

Для графового моделювання інформації та її зберігання використовується база даних Neo4j, що є оптимізованою для обходу великих графів, забезпечуючи високу продуктивність графових запитів і пристосованість до застосування в додатках, які потребують розуміння складних взаємозв’язків між бізнес-об’єктами (наприклад, рекомендаційні системи та фінансові застосунки). Neo4j використовує Cypher, потужну та інтуїтивно зрозумілу мову запитів, що дозволяє легко шукати потрібну інформацію серед комплексної структури графів. Також Neo4j легко масштабується по горизонталі за допомогою сегментування та реплікації даних.

Графові нейронні мережі було створено за допомогою бібліотеки PyG. Вона містить реалізацію сучасних моделей GNN та стандартних графових

наборів даних, надаючи інструменти для експериментування з ними. PyG побудована на основі бібліотеки машинного навчання PyTorch, якій притаманна гнучкість та простота використання. Це дозволяє розробникам використовувати інструменти PyTorch для навчання та розгортання моделей.

У ролі великої мовної моделі була обрана GPT-4o від OpenAI, що на цей момент є найпотужнішою з широко доступних моделей обробки природної мови. Особливістю цієї LLM є можливість надання структурованих відповідей у вигляді JSON, що може бути переданий на вхід традиційних функцій для отримання відповіді на запит користувача. Також GPT-4o може інтерпретувати зображення та аудіо для надання мультимодальних можливостей розробленій системі.

3.1 Архітектура розробленої системи підтримки бізнес-рішень

Архітектура графової системи підтримки ухвалення рішень інтегрує декілька компонентів для забезпечення комплексного рішення для аналізу бізнес-даних. Основними компонентами є:

- база даних Neo4j, що використовується для зберігання та управління графічними даними; дозволяє ефективно робити запити та візуалізувати взаємозв’язки між різними точками даних;
- графова нейронна мережа PyG; що застосовується для розв'язання задач класифікації вузлів та прогнозування зв’язків;
- велика мовна модель OpenAI, що використовується для знаходження найбільш доцільних інструментів для активації на основі запиту від користувача та для представлення результатів використання цих інструментів у зручному форматі;
- сервер Flask, що надає зв’язок з базою даних Neo4j, графовою нейронною мережею PyG та великою мовною моделлю OpenAI;

API містить кінцеві точки для CRUD-операцій над вузлами та зв’язками, обробки завантаження файлів, створення транскрипцій, доповнень тексту, автентифікації та виконання функцій для передачі великій мовній моделі.

Продемонструємо потік інформації в розробленій системі на конкретному прикладі. Нехай співробітник певної компанії хоче дізнатися, чи є якісь із нещодавніх платежів на користь цього бізнесу підозрілими на шахрайство. Взаємодія співробітника компанії із системою зображена на рис. 3.1.

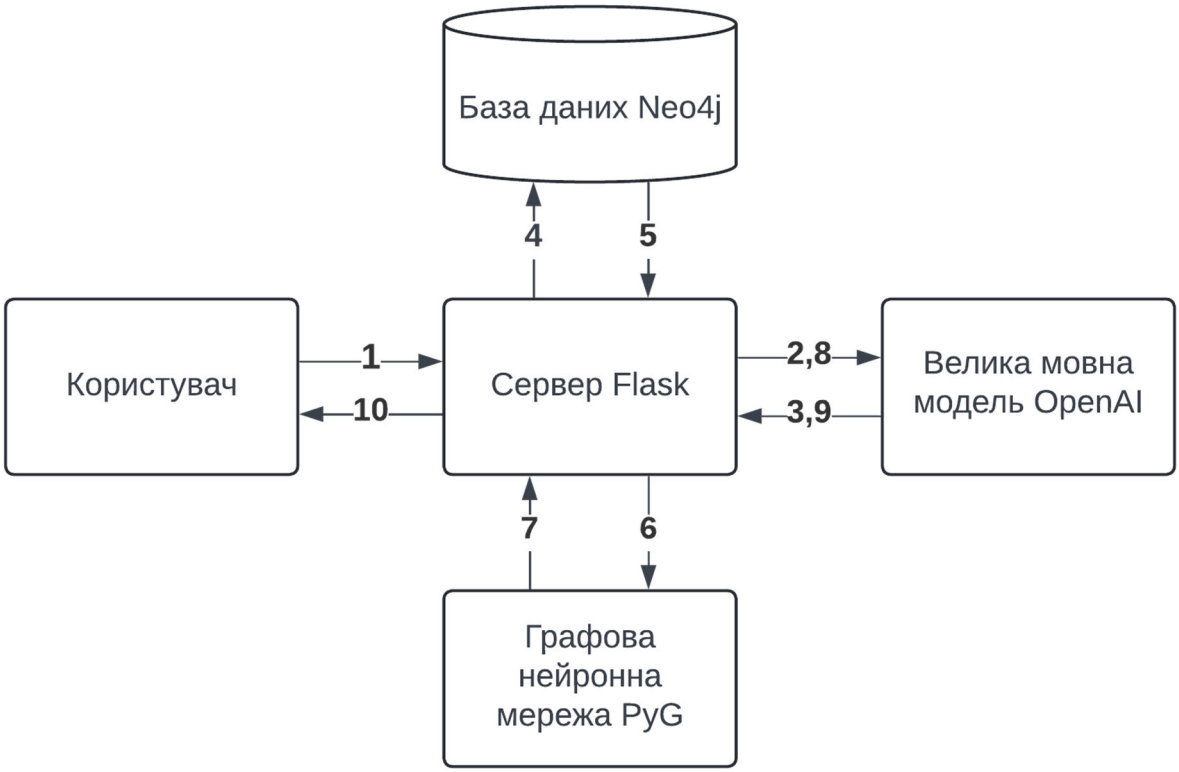

Рисунок 3.1 – Архітектура розробленої графової системи підтримки ухвалення рішень у бізнесі

1. Користувач взаємодіє з системою через інтерфейс (UI), надсилаючи певний запит, наприклад: «Чи є останні платежі підозрілими на шахрайство?».

2. Сервер Flask перенаправляє запит користувача до великої мовної моделі OpenAI, щоб отримати розуміння контексту, наприклад, як цей запит пов’язаний із доступними інструментами.
3. Велика мовна модель OpenAI інтерпретує запит користувача, визначає доцільні інструменти для використання і повертає їх перелік на сервер Flask, наприклад функцію «отримати останні платежі» чи «класифікувати платіж».
4. Сервер Flask робить запит у базу даних Neo4j для отримання релевантних даних графа, пов’язаних із запитом користувача, наприклад, вузли з міткою «транзакція» та зв’язки між ними.
5. База даних Neo4j виконує запит і повертає відповідні вузли та зв’язки на сервер Flask.
6. Якщо функція не потребує залучення графової нейронної мережі, наприклад, функція «отримати останні платежі», то переходимо до кроку 8. В іншому разі сервер Flask надсилає дані до графової нейронної мережі PyG для розв'язання задач машинного навчання, наприклад, класифікація вузлів як «справжній» чи «шахрайський».
7. Мережа PyG обробляє дані та повертає прогнози на сервер Flask, наприклад, ймовірність того, що певна транзакція пов’язана з шахрайством.
8. Сервер Flask надсилає прогнози у велику мовну модель OpenAI для інтерпретації результатів, наприклад: «ймовірність класу 0 дорівнює 0.9, ймовірність класу 1 дорівнює 0.1». Якщо модель визначила, що відповідь на питання користувача отримано, то вона надсилає кінцевий результат на сервер Flask. В іншому разі переходимо до кроку 3.
9. Велика мовна модель обробляє результати для представлення їх у більш зрозумілому для користувача вигляді, наприклад, перетворює відповідності ключ-значення в об’єкті JSON на текст.
10. Сервер Flask агрегує інформацію з бази даних, графової нейронної

мережі та великої мовної моделі, форматує відповідь та надсилає кінцевий результат користувачу, наприклад: «Останні транзакції з великою ймовірністю не є шахрайськими».

Файлову структуру системи можна поділити на такі рівні:

- рівень серверних даних, що містить кеш для миттєвої обробки повторюваних запитів та перелік типових наборів бізнес-даних (наприклад, AmazonComputers), що було завантажено у базу даних для подальшої обробки;
- рівень бази даних, що реалізує підключення до бази даних Neo4j та виконання запитів для ефективного пошуку та маніпулювання даними;
- рівень моделей, що містить реалізацію графової згорткової мережі (GCN), навчені моделі (наприклад, класифікатор транзакцій) та допоміжні функції (наприклад, отримання останніх транзакцій та передбачення класу транзакції); також включає блокноти Jupyter для попередньої обробки даних та розміщення у базі даних (data_to_neo4j.ipynb) та для отримання інформації з бази даних і навчання моделей (neo4j_to_ml.ipynb);
- рівень інструментів, що містить функції, які активуються за допомогою великої мовної моделі для відповіді на питання користувача;
- рівень маршрутів, що реалізує шляхи для обробки різних типів запитів, наприклад, автентифікація користувачів, файлові операції, запити до графової бази даних, взаємодія з інструментами;
- рівень статичних даних, що містить файли клієнтської частини системи, наприклад, шаблони, стилі, скрипти, зображення;
- рівень утиліт, що містить допоміжні функції, конфігурацію, логування, тестування тощо.

Після запуску системи відбувається створення індексів у базі даних. Поля ідентифікатору, назви та дати створення запису отримують стандартний індекс

для збільшення швидкості операцій за цими полями, причому на ідентифікатор накладається умова унікальності. Поле embedding отримує векторний індекс із косинусною функцією подібності.

Для підрахування embeddings вузлів використовується комбінований підхід. Назва, вміст та мітки вузла конкатенуються та передаються у модель text-embedding-3-small від OpenAI, що знаходить вектор розмірності 1536. Ознаки на рівні вузлів, як-от степінь, центральність та PageRank комбінуються з цим вектором для надання інформації про структуру графа. Отриманий вектор надалі використовується графовою нейронною мережею для уточнення представлення вузлів. Якщо був активований перемикач «AI insights», то велика мовна модель отримує доступні інструменти та повертає список активованих функцій.

3.2 Створення графових нейронних мереж для прогнозування бізнес-даних

Для створення графових нейронних мереж були обрані набори даних KarateClub, Cora та AmazonComputers, що значно варіюються у розмірі та представляють різні аспекти ухвалення бізнес-рішень. Така різноманітність даних дозволила провести комплексний аналіз ефективності розробленої системи. Структура цих наборів даних зображена у табл. 3.1.

Таблиця 3.1 – Структура обраних наборів даних

Назва	*К-ть вузлів*	*К-ть зв'язків*	*К-ть ознак*	*К-ть класів*
KarateClub	34	156	34	4
Cora	2708	10 556	1433	7
AmazonComputers	13 752	491 722	767	10

Завантаження наборів даних у графову базу даних Neo4j відбувається у файлі data_to_neo4j.ipynb. Усі дані потрапляють у Neo4j не напряму, а за допомогою розробленого API, що дозволяє перевірити коректність усіх компонентів системи. Великий розмір вхідних даних зумовлює необхідність використання кінцевих точок API, які підтримують пакетне завантаження (/api/nodes/batch та /api/edges/batch). Великі значення розміру пакетів надають можливість використання внутрішньої оптимізації об'ємних запитів, що проводяться базою даних Neo4j, а маленькі – вищу стабільність системи шляхом передавання меншої кількості інформації мережею. Для надсилання пакетних даних у систему був обраний розмір пакетів 10 000 як компроміс між швидкістю завантаження та стабільністю системи. Перед надсиланням даних відкривається сесія HTTP для підтримання з'єднання після автентифікації. У ході надсилання даних проводиться збереження відповідностей оригінальних ідентифікаторів елементів набору даних та нових ідентифікаторів, що генеруються системою шляхом конкатенації головної мітки вузла та невеликого випадково згенерованого рядка для забезпечення унікальності та простого сприйняття цього рядка у складі посилання на сторінку. Відповідності записуються у файли з розширенням .jsonl, що дозволяє використовувати кожний рядок файлу як окремий JSON і не завантажувати цілий файл у пам'ять системи при роботі з цими даними.

Після завантаження набору даних у базу даних Neo4j виконуються зворотні дії для перевірки коректності роботи кінцевих точок отримання даних (/api/nodes та /api/edges). Отримуються пакети вузлів та зв'язків розміром 10 000, після чого проводиться попередня обробка для зведення у формат, що використовується бібліотеками PyTorch та PyG для навчання графової нейронної мережі. Оброблені дані записуються у файл .pkl, що є стандартним способом збереження стану об'єкту в Python для подальшого використання.

Класифікація даних відбувається за допомогою класу GNNClassifier, що може імплементувати стандартні архітектури графових нейронних мереж: GCN, GAT, Graph Transformer тощо. Проводиться навчання відповідної мережі,

після чого її ваги зберігаються у файлі gnn_classifier.pth.

Вхідні дані розбиваються на окремі частини: тренувальна множина використовується для навчання мережі, валідаційна – для моніторингу процесу навчання та порівняння мереж з різними гіперпараметрами між собою, тестова – для остаточної оцінки обраної мережі. У випадках, коли відповідний поділ не був наданий авторами наборів даних, ці дані було розбито у співвідношенні $0.70 - 0.15 - 0.15$. З огляду на те, що класи можуть бути незбалансованими, знаходиться стратифікована проба, тобто розбиття йде не на рівні загальної популяції, а на рівні кожного класу.

У ході навчання зберігаються значення функції втрат та метрики для подальшої візуалізації процесу тренування мережі. Також було використано стратегію ранньої зупинки: якщо ефективність моделі на валідаційному наборі не покращується протягом 10 епох, то тренування припиняється.

Для підбору найкращих гіперпараметрів мережі була використана бібліотека Optuna. У ній процес знаходження гіперпараметрів відбувається в межах окремого «дослідження», що розбито на «випробування». Ціллю кожного дослідження є максимізація певної метрики. Простір пошуку зображений у табл. 3.2.

Таблиця 3.2 – Простір пошуку гіперпараметрів мережі

Гіперпараметр	*Тип*	*Мінімум*	*Максимум*
Кількість шарів	integer	2	4
Кількість прихованих нейронів	integer	16	64
Частка dropout	float	0.0	0.5
Швидкість навчання	float	10^{-4}	10^{-2}

У тих випадках, коли присутня незбалансованість класів, їхні ваги модифікуються так, щоб збільшити важливість класу з меншою кількістю прикладів. Моделі використовують Adam як оптимізатор, перехресну ентропію як функцію втрат, macro F_1-score як метрику.

Серед розглянутих шарів: Linear, GCNConv, GATConv та

TransformerConv, що були описані в другому розділі роботи. Кожний тип шару оцінюється упродовж 10 випробувань за допомогою випадкового вибору комбінації гіперпараметрів, складання мережі з використанням копій цього шару, між якими стоїть активаційна функція ReLU та шар регуляризації Dropout, а в кінці – шар Linear, що перетворює отримані embeddings на класи. Навчання отриманої мережі відбувається на тренувальному наборі, а оцінка параметра macro F_1-score – на валідаційному наборі.

Після закінчення підбору гіперпараметрів проводиться візуалізація ефективності створених моделей за кожним типом шару за допомогою діаграми розмаху. Далі створюються лінійні графіки історії тренування найкращої моделі, що зображають значення функції втрат та метрики з плином епох. Також відображається матриця невідповідностей, рядки якої – це прогнозований клас, стовпці – справжній клас, а на перетині стоїть відповідна кількість прикладів. Наприкінці створюється порівняльна таблиця найкращих моделей за кожним типом шару, що містить влучність, повноту та macro F_1-score для кожного набору даних.

Набір даних KarateClub

KarateClub [42] містить вузли осіб і зв’язки між ними. Його можна використати для вивчення соціальної динаміки, формування спільнот в компаніях, визначення ключових осіб або аналізу командних структур для кращого управління проєктами. Цей набір даних був створений з метою знаходження джерела конфліктів, тому він може допомогти в розробці стратегій для їхнього пом’якшення і збільшення продуктивності команд. KarateClub представляє особливу складність через невелику тренувальну вибірку: на кожний з чотирьох класів існує лише один тренувальний екземпляр, тому важливо використати усю наявну інформацію про структуру графа для досягнення прийнятних показників. Результати тренування графових нейронних мереж на наборі KarateClub зображено на рис. 3.2–3.5 і табл. 3.3.

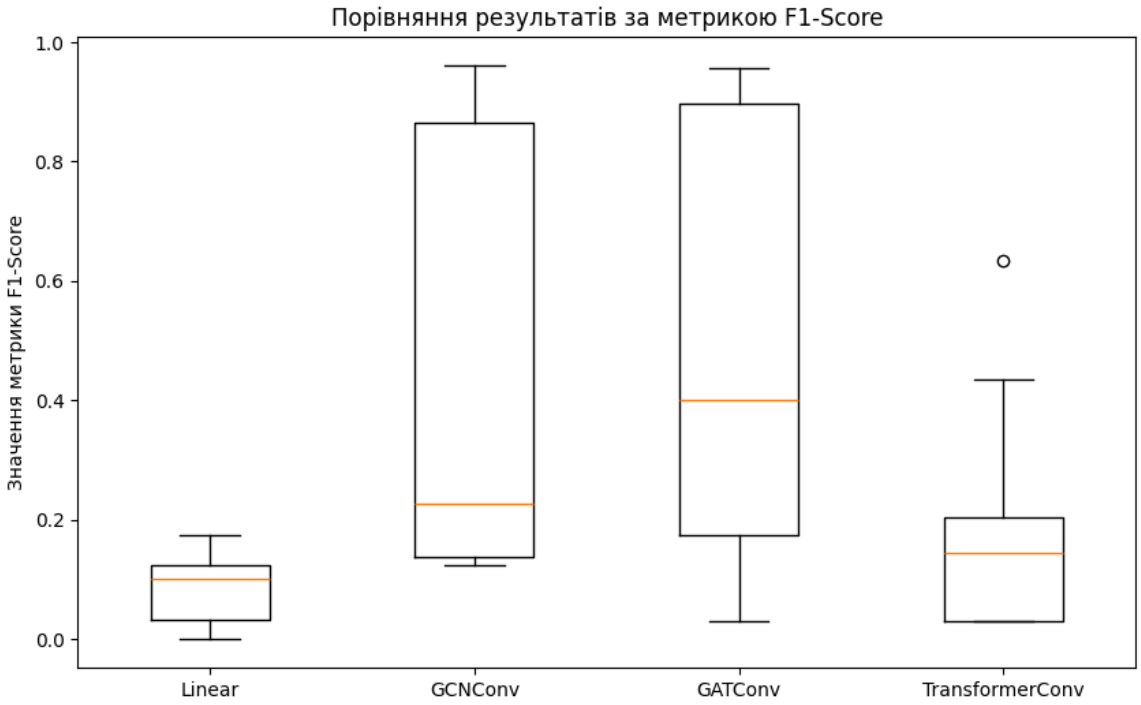

Рисунок 3.2 – Діаграма розмаху для набору даних KarateClub

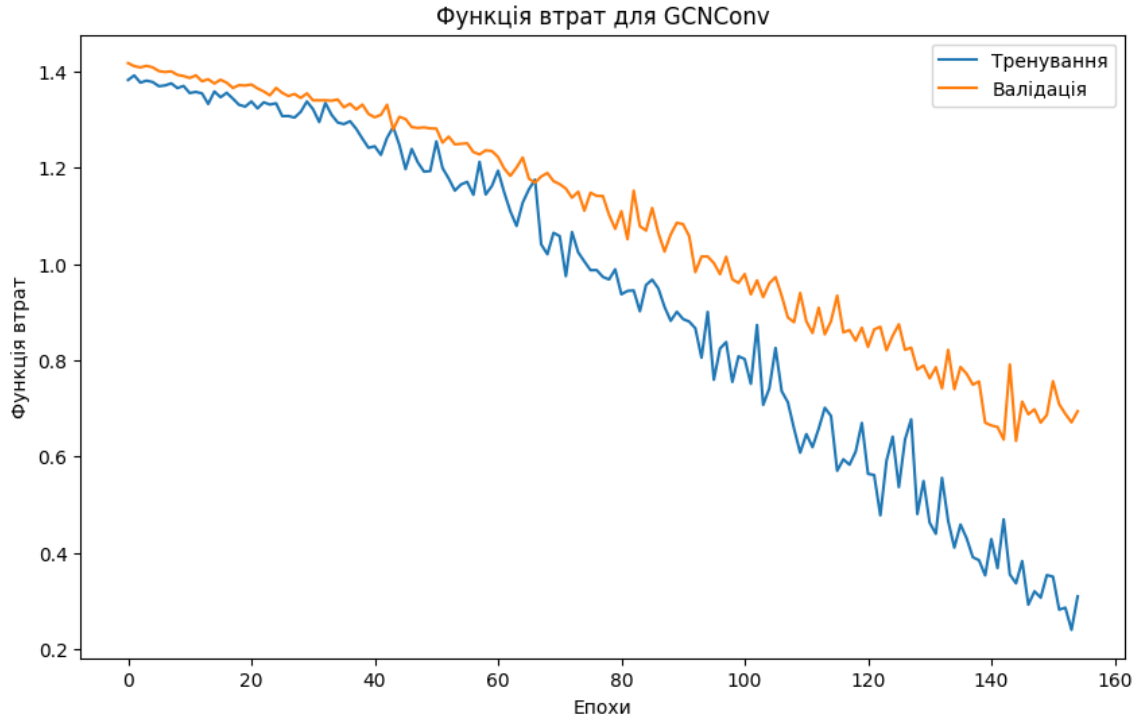

Рисунок 3.3 – Функція втрат найкращої моделі для набору даних KarateClub

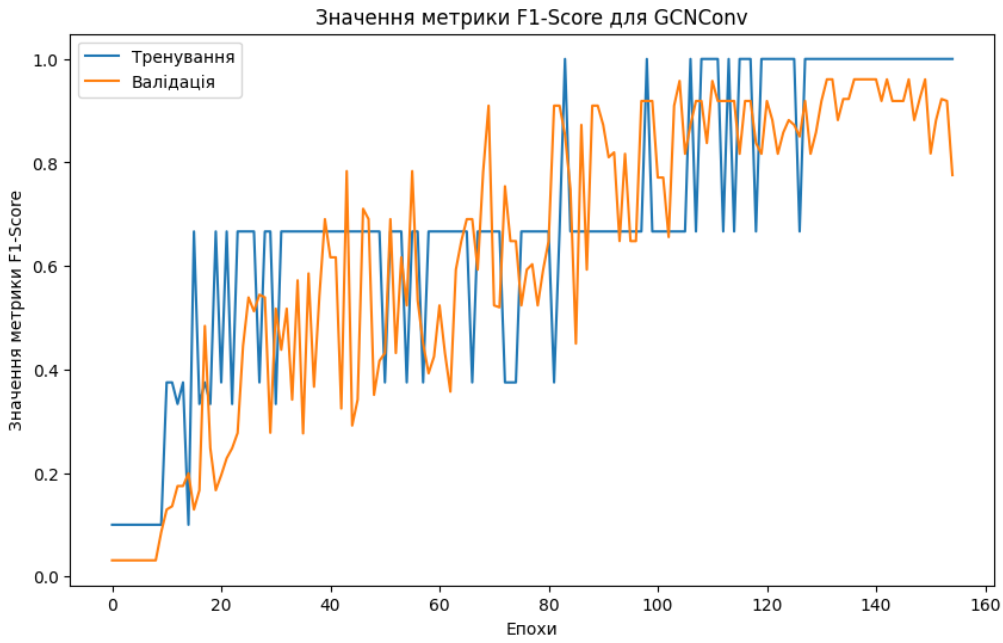

Рисунок 3.4 – Значення метрики macro F1 найкращої моделі для набору даних KarateClub

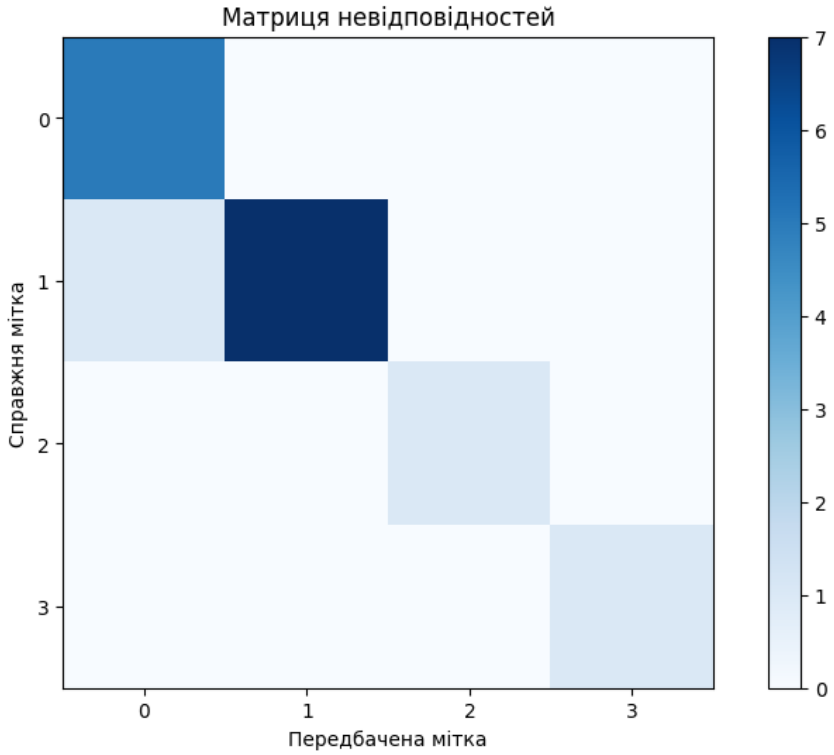

Рисунок 3.5 – Матриця невідповідностей найкращої моделі для набору даних KarateClub

Таблиця 3.3 – Порівняння метрик для набору даних KarateClub

Тип шару	*Влучність*	*Повнота*	*Метрика F_1*
Linear	0.133333	0.25000	0.173913
GCNConv	0.958333	**0.96875**	**0.960606**
GATConv	**0.972222**	0.95000	0.957516
TransformerConv	0.637500	0.82500	0.633333

Отримані результати характеризуються великою дисперсією між метриками моделей, що може бути пояснено невеликою кількістю вузлів та зв'язків між ними. Найкращим типом шарів для цієї задачі стали згорткові, при цьому вони значно випередили шари лінійних перетворень, тобто інформація про зв'язки між особами значно допомагає моделі у виконанні прогнозування. Тренування мережі виявилося нестабільним, а значення метрики на тестовому наборі дорівнює 0.6417, що вказує на помірні узагальнювальні здібності моделі.

Набір даних Cora

Cora [43] складається з вузлів наукових статей і цитувань між ними. Цей набір допомагає виявити впливові статті та ключові дослідницькі тенденції, що мають відношення до бізнесу. Аналізуючи діаграми цитувань, компанії можуть виявити нових стратегічних партнерів і потужних конкурентів. Cora дає можливість зрозуміти структуру мереж інновацій, визначаючи центральні вузли, що пов'язують різні дослідницькі напрямки. Така спрямованість є цінною для отримання інформації про міждисциплінарну співпрацю. Також аналіз статей може допомогти у пошуку масштабованих та ефективних рішень, що можуть бути застосовані в реальному світі для покращення продуктів, якими користуються мільйони клієнтів. Цьому набору притаманна велика кількість вузлів та зв'язків, тому графові мережі можуть отримати багато цінної інформації для виконання точних прогнозів. Результати тренування графових нейронних мереж на наборі Cora зображено на рис. 3.6–3.9 і табл. 3.4.

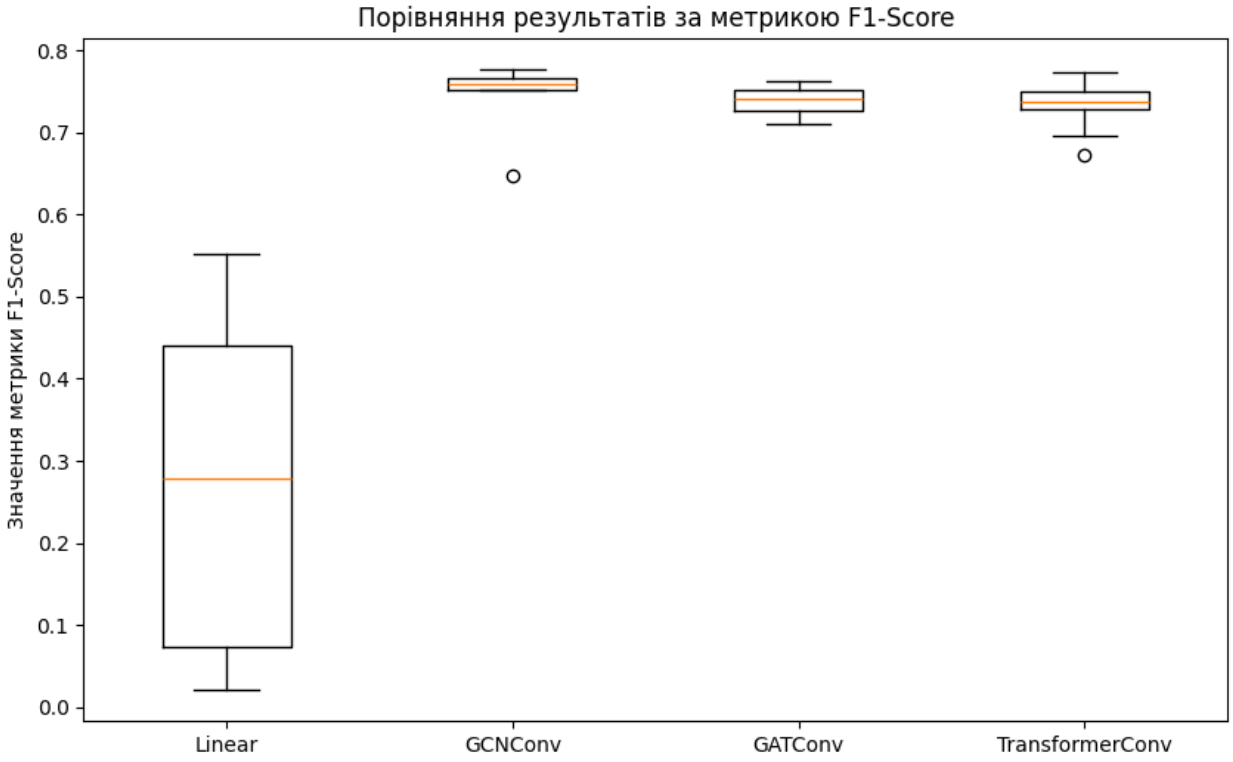

Рисунок 3.6 – Діаграма розмаху для набору даних Cora

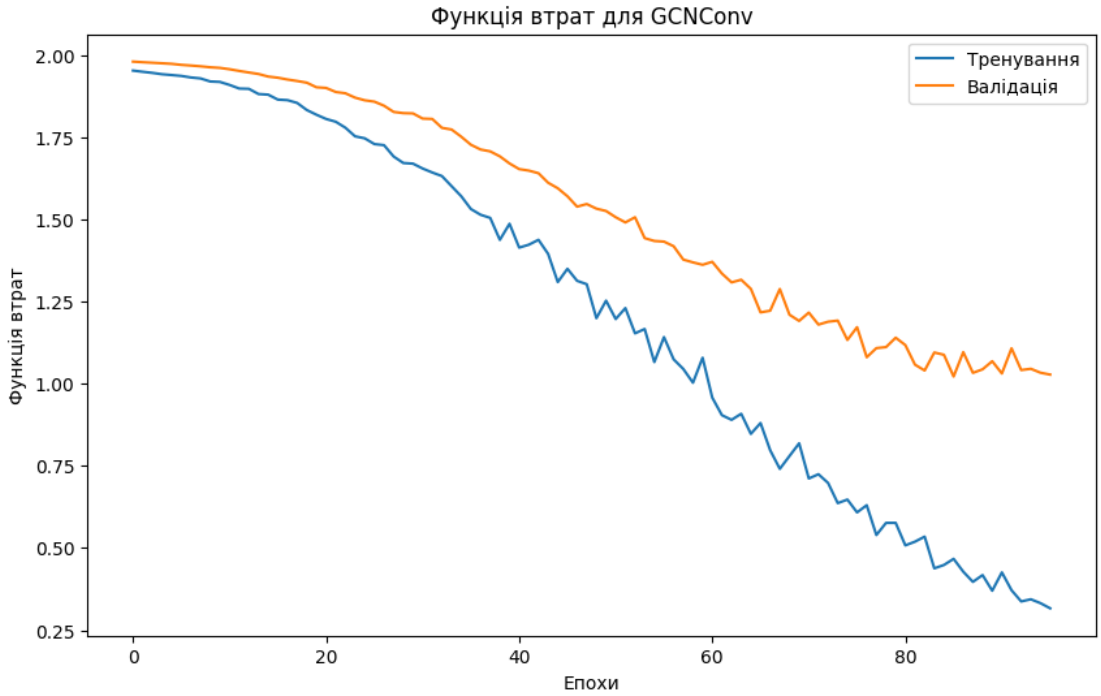

Рисунок 3.7 – Функція втрат найкращої моделі для набору даних Cora

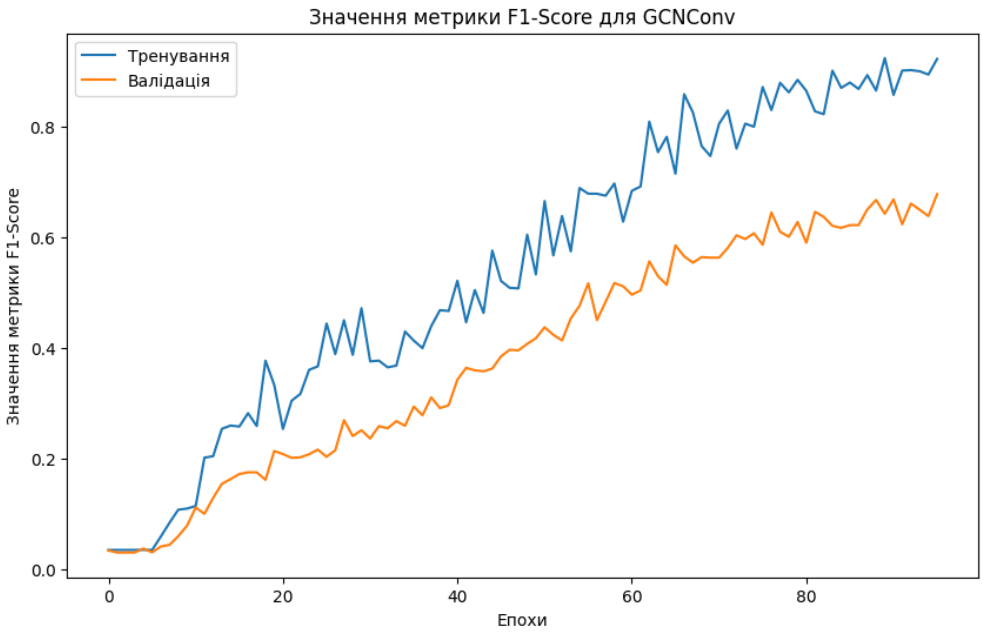

Рисунок 3.8 – Значення метрики macro F1 найкращої моделі для набору даних Cora

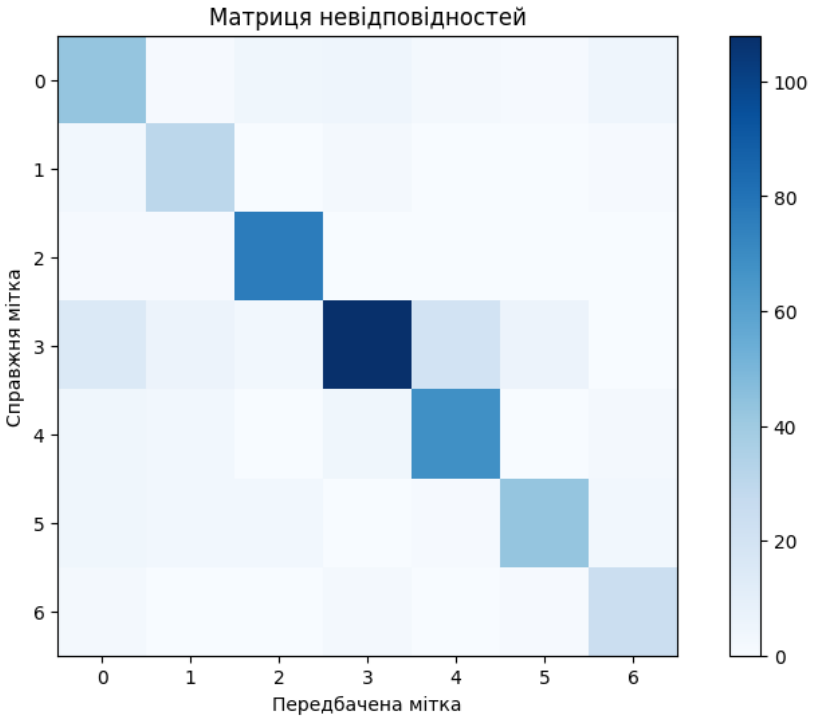

Рисунок 3.9 – Матриця невідповідностей найкращої моделі для набору даних Cora

Таблиця 3.4 – Порівняння метрик для набору даних Cora

Тип шару	*Влучність*	*Повнота*	*Метрика F_1*
Linear	0.567055	0.569349	0.551925
GCNConv	**0.761633**	**0.802519**	**0.776376**
GATConv	0.749743	0.790887	0.761816
TransformerConv	0.760167	0.801861	0.773919

Отримані результати свідчать про високу ефективність графових шарів на цьому наборі даних у порівнянні з шарами лінійних перетворень, тобто інформація про посилання на інші статті значною мірою допомагає у класифікації цих статей. Усі графові мережі характеризуються стабільністю навчання незалежно від обраних гіперпараметрів, при цьому модель на основі GCNConv трохи випереджає інші графові моделі та має показник 0.7828 на тестовому наборі даних, що свідчить про високі узагальнювальні властивості обраної моделі.

Набір даних AmazonComputers

AmazonComputers [44] містить вузли товарів, а зв’язки між ними існують, коли ці товари часто купують разом. Ознаками вузлів є перетворені відгуки користувачів про товари. Цей набір може застосовуватися для аналізу думок клієнтів з метою покращення товарів компанії, обслуговування клієнтів та розробки цільових маркетингових стратегій. Розуміння того, які товари найчастіше купують разом, дає змогу надавати персоналізовані рекомендації, підвищуючи рівень задоволеності клієнтів і збільшуючи продажі. Великий розмір і різноманітність набору даних дають багато корисної інформації нейронним мережам, але вимагають потужних обчислювальних ресурсів та ефективних алгоритмів для проведення аналізу. Крім того, споживчі вподобання та ринкові тенденції змінюються з часом, що зумовлює виконання прогнозування в реальному часі. Результати тренування графових нейронних мереж на наборі AmazonComputers зображено на рис. 3.10–рис. 3.13 і табл. 3.5.

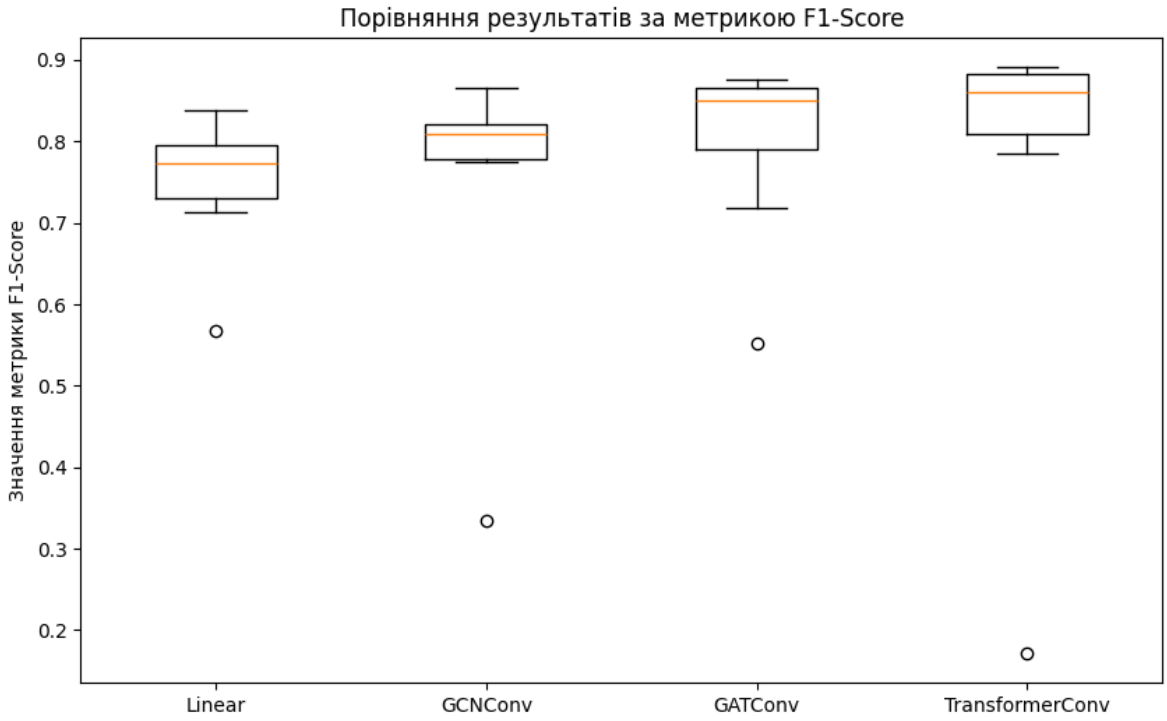

Рисунок 3.10 – Діаграма розмаху для набору даних Amazon

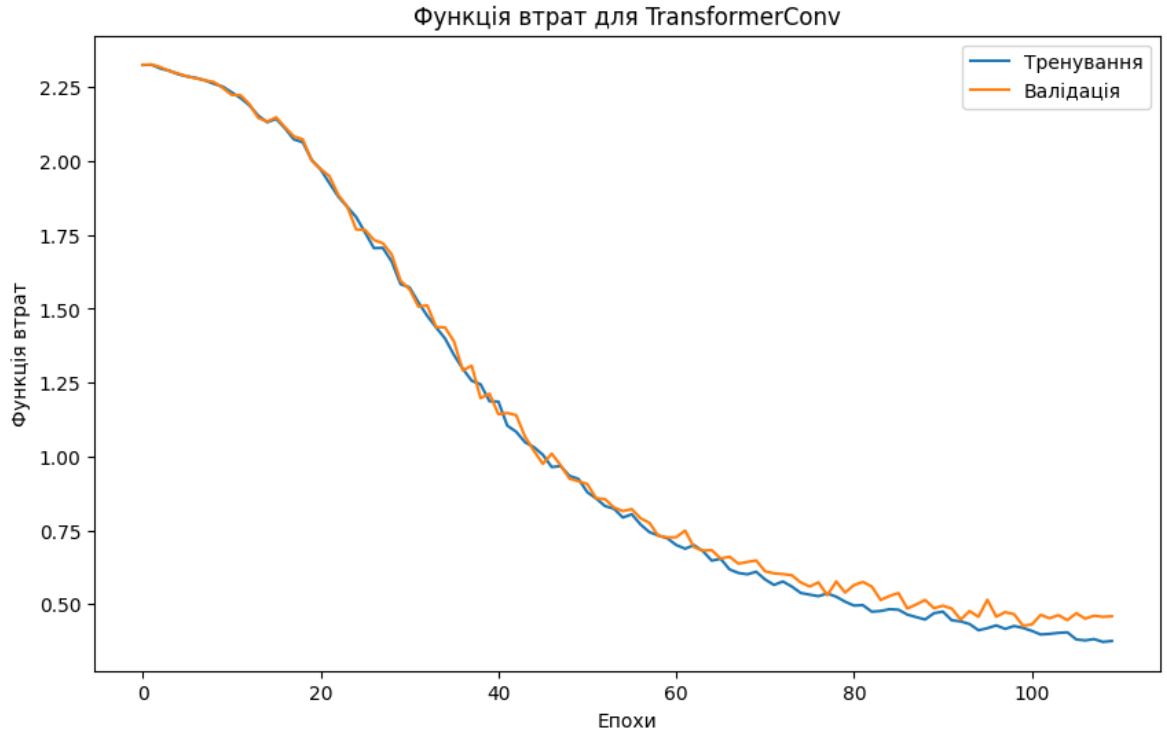

Рисунок 3.11 – Функція втрат найкращої моделі для набору даних Amazon

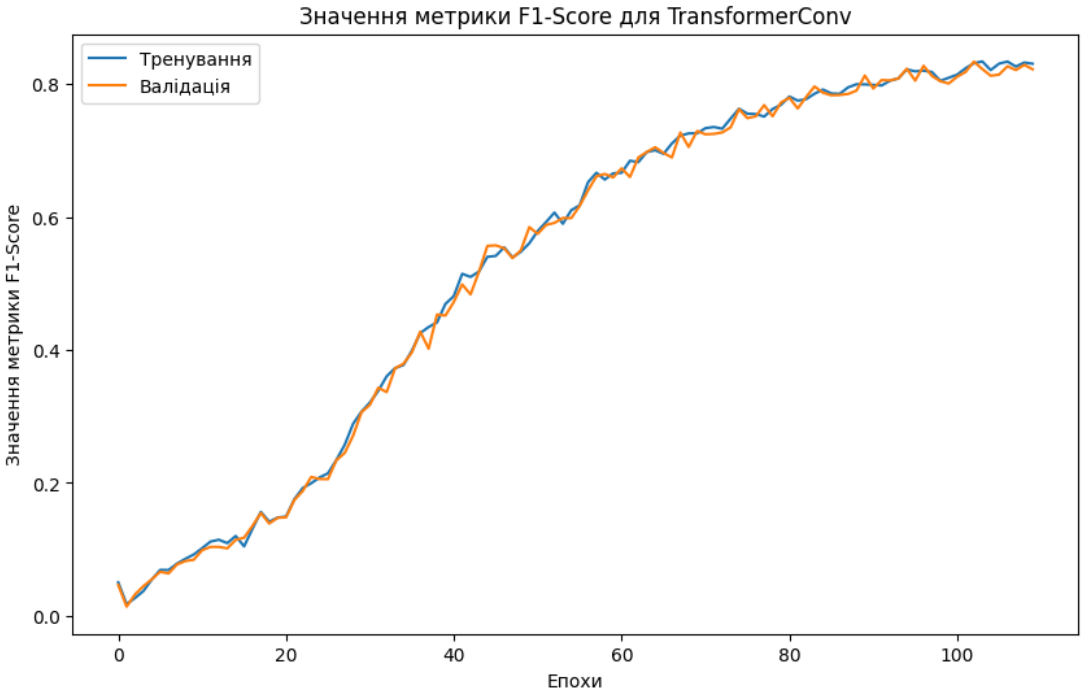

Рисунок 3.12 – Значення метрики macro F1 найкращої моделі для набору даних Amazon

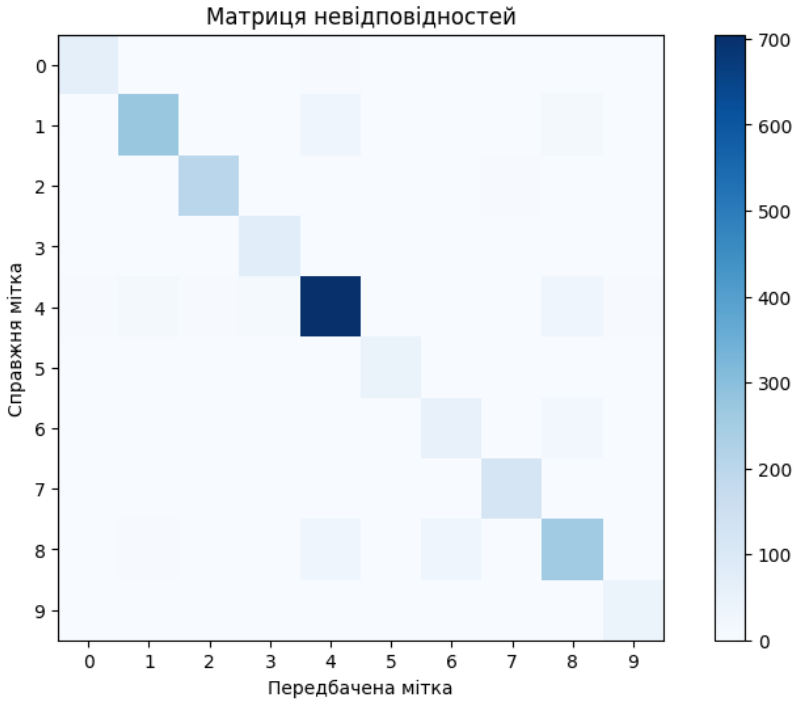

Рисунок 3.13 – Матриця невідповідностей найкращої моделі для набору даних Amazon

Таблиця 3.5 – Порівняння метрик для набору даних Amazon

Тип шару	*Влучність*	*Повнота*	*Метрика F_1*
Linear	0.807372	0.876922	0.837431
GCNConv	0.835573	**0.914367**	0.865132
GATConv	0.851706	0.910270	0.876351
TransformerConv	**0.878475**	0.904881	**0.890691**

Отримані результати свідчать про невелику перевагу графових шарів над шарами лінійних перетворень, тобто інформація про товари, які часто купляють разом, допомагає у класифікації цих товарів. Тренування мереж є стабільним і не має великої залежності від обраних гіперпараметрів. Шари мережі-трансформера виявилися найбільш ефективними та мають показник 0.8952 на тестовому наборі даних, що вказує на дуже високі узагальнювальні властивості обраної моделі.

3.3 Взаємодія з розробленою системою через API

Інтерфейс програмування застосунку (API) є важливою частиною створеної системи та додає можливість простої інтеграції із зовнішніми сервісами. Це дозволяє розвинути екосистему навколо цього додатку, заохочуючи інших розробників створювати свої продукти на його основі і розширювати його охоплення та функціональність. API дає можливість монетизувати дані або послуги бізнесу, пропонуючи їх іншим компаніям на платній основі. Також API забезпечує рівень абстракції, що дозволяє вносити зміни у внутрішню логіку, не впливаючи на зовнішній інтерфейс.

API побудований на основі REST-архітектури, що забезпечує простоту використання та масштабованість. Кожна кінцева точка представляє конкретний ресурс системи, а взаємодія з ними здійснюється за допомогою

стандартних HTTP-методів: GET, POST, PUT, DELETE. Кінцеві точки API представлені у табл. 3.6, параметри – у табл. 3.7, коди стану – у табл. 3.8.

Таблиця 3.6 – Опис кінцевих точок API

Кінцева точка	*Метод*	*Опис*	*Параметри*	*Коди стану*
/api/labels	GET	Отримання списку доступних позначок вузлів	–	200 500
/api/nodes/:node_id	GET	Отримання інформації про вузол	node_id	200 400 401 500
	DELETE	Видалення вузла		
/api/nodes	GET	Отримання інформації про список вузлів	page page_size start_date end_date sort_by sort_order q filter	200 400 500
	POST	Створення вузла	labels title content	201/200 400 500
	PUT	Зміна інформації про вузол		
/api/nodes/batch	POST	Створення групи вузлів	labels title content	201 400 401 500
/api/edges/:edge_id	GET	Отримання інформації про зв’язок	edge_id	200 400 500
/api/edges	GET	Отримання інформації про список зв’язків	page page_size start_date end_date sort_by sort_order filter	200 400 500
	POST	Створення зв’язку	src dst type	201 400 401 500

Продовження табл. 3.6

Кінцева точка	*Метод*	*Опис*	*Параметри*	*Коди стану*
/api/edges/batch	POST	Створення групи зв’язків	src dst type	201 400 401 500
/api/files/:file_id	GET	Отримання інформації про файл	file_id	200 400 401 500
	DELETE	Видалення файлу		
/api/files	POST	Створення файлу	file	201 400 500
/api/audio/transcriptions	POST	Створення транскрипції аудіофайлу	file	200 400 401 500
/api/chat/completions	POST	Створення доповнень тексту	messages: role content	200 400 401 500
/api/login	POST	Вхід у систему	username password	200 400 401 404 500
/api/signup	POST	Реєстрація в системі	username password	201 400 409 500
/api/logout	POST	Вихід із системи	–	200 401 500
/api/tools	GET	Отримання списку доступних функцій	–	200 400 401 500
	POST	Використання доступної функції	name arguments	
/	GET	Перегляд головної сторінки	–	200 500
/item/:item_id	GET	Перегляд об’єкта	item_id	200 500

Закінчення табл. 3.6

Кінцева точка	*Метод*	*Опис*	*Параметри*	*Коди стану*
/search	GET	Пошук найбільш схожих вузлів до запиту з помічником на основі ШІ	page page_size start_date end_date sort_by sort_order q filter	200 500
/new	GET	Перегляд сторінки створення нового вузла	–	200 401 500

Таблиця 3.7 – Опис параметрів API

Назва	*Тип*	*Опис*
node_id/edge_id/file_id	string	Ідентифікатор вузла/зв’язку/файлу
page	integer	Номер поточної сторінки
page_size	integer	Кількість записів на кожній сторінці
start_date/end_date	string	Початкова/кінцева дата у форматі YYYY-MM-DDTHH:MM:SSZ для фільтрації за датою створення запису
sort_by	string	Поле, за яким виконується сортування
sort_order	string	Порядок сортування (ASC/DESC)
q	string	Пошуковий запит
filter	object	Відповідність ключ–значення для фільтрації за значенням
labels	array	Мітки вузла
title/content	string	Назва/опис вузла
src/dst	string	Ідентифікатор початкового/кінцевого вузлів, між якими існує зв’язок
type	string	Тип зв’язку
file	form-data	Файл зображення, аудіозапису, документа тощо
messages	array	Список повідомлень, що складають історію діалогу з мовною моделлю; мають атрибути ролі співрозмовника та змісту
username/password	string	Ім’я/пароль користувача
name	string	Назва використаного інструмента
arguments	object	Відповідність ключ–значення для передачі аргументів у інструмент

Таблиця 3.8 – Опис кодів стану API

Код	*Використання*
200 OK	Загальне позначення успішної операції
201 Created	Успішне створення вузла/зв’язку/файлу
400 Bad Request	Некоректний запит, наприклад, спроба фільтрації за відсутніми полями
401 Unauthorized	Спроба виконати дію з обмеженим доступом, коли не було здійснено вхід у систему; наприклад, створення/зміна/видалення вузла/зв’язку/файлу
404 Not Found	Спроба отримати відсутній ресурс, наприклад, вузол/зв’язок/файл
409 Conflict	Спроба створити дублікат наявного ресурсу, наприклад, запис користувача з повторюваним ім’ям
500 Internal Server Error	Загальне позначення помилки на стороні сервера

3.4 Взаємодія з розробленою системою через інтерфейс користувача

Для забезпечення інтуїтивної роботи з системою був розроблений користувацький інтерфейс, що містить такі елементи:

- пошуковий рядок, що дозволяє користувачам робити запити до бази даних природною мовою; під рядком пошуку система надає пропоновані мітки та часові проміжки, в яких можуть знаходитися потрібні дані;
- перемикач «AI insights»: якщо увімкнений, система використовує велику мовну модель для інтерпретації запиту користувача та запуску відповідних функцій, в іншому разі відбувається пошук лише за найближчими embeddings до пошукового рядка;
- секція «AI insights» (рис. 3.14) містить перелік інструментів, що було активовано, з відповідними назвами та аргументами; у кінці секції відбувається підсумовування результатів великою мовною моделлю для полегшеного сприйняття та підтримки ухвалення

обґрунтованих рішень в бізнесі;

– результати пошуку (рис. 3.15) містять перелік найкращих збігів за embedding, що задовольняють застосовані фільтри та тип сортування; кожний результат складається з назви об’єкта, посилання на його сторінку, його міток, розширеного опису, міри схожості на пошуковий запит та дати створення;
– спливаючі меню реєстрації акаунта та входу в акаунт (рис. 3.16) містять поля імені користувача та пароля;
– візуалізація знайдених об’єктів у вигляді вузлів графа (рис. 3.17), між якими відтворюються наявні зв’язки;
– екран ручного створення вузла з редактором Trix, що підтримує форматування тексту та завантаження файлів; причому зображення та аудіо перетворюються на текст (рис. 3.18–рис.3.19).

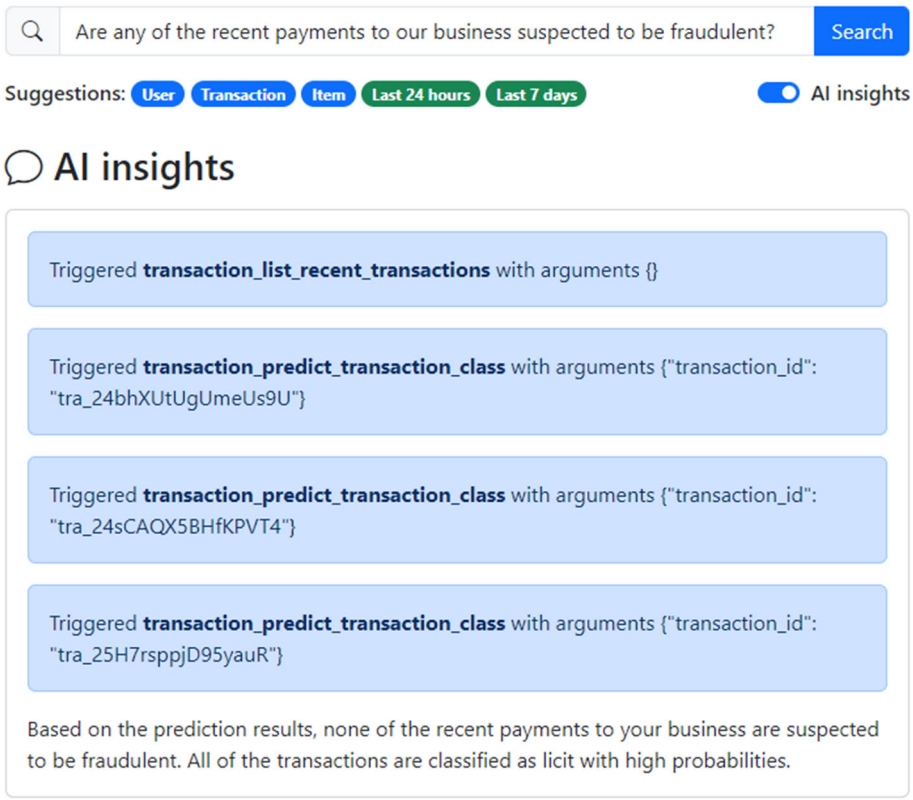

Рисунок 3.14 – Пошук відповіді на питання користувача за допомогою великої мовної моделі

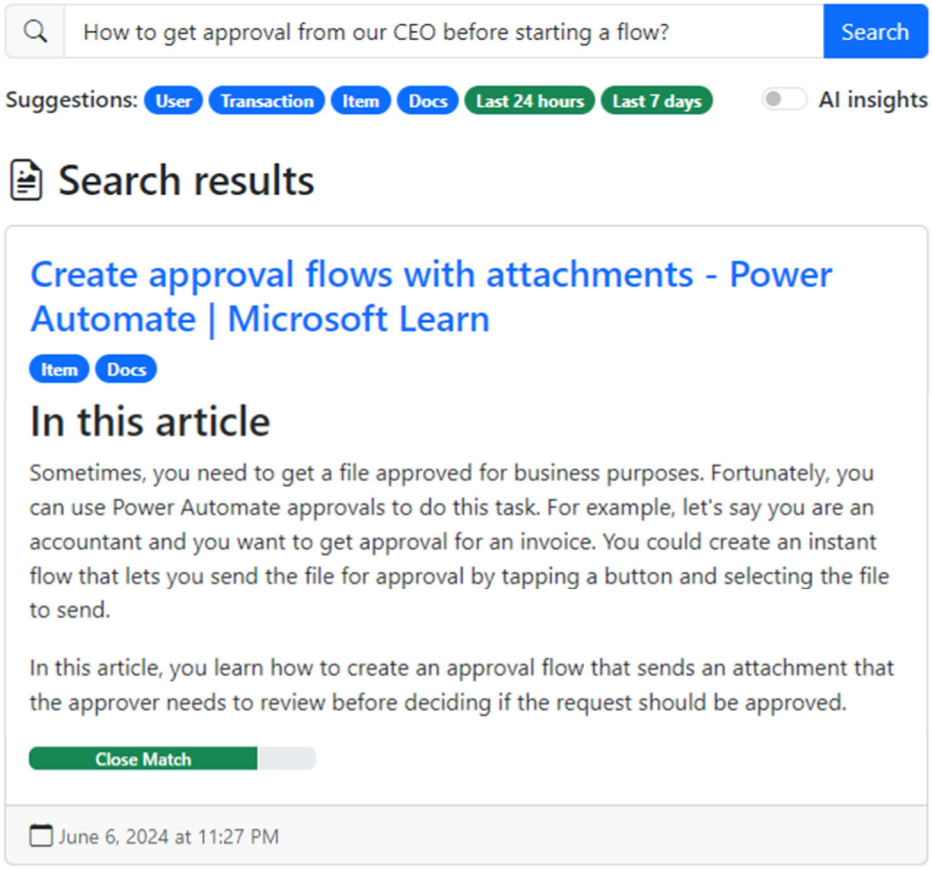

Рисунок 3.15 – Пошук відповіді на питання користувача за допомогою найближчих embeddings

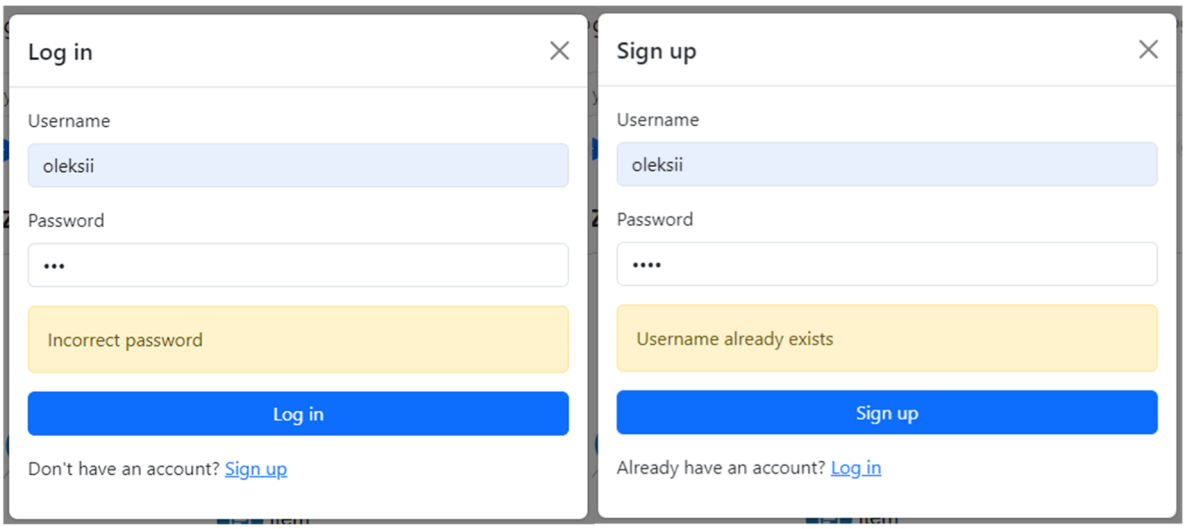

Рисунок 3.16 – Екрани входу та реєстрації містять перевірку коректності введеної інформації

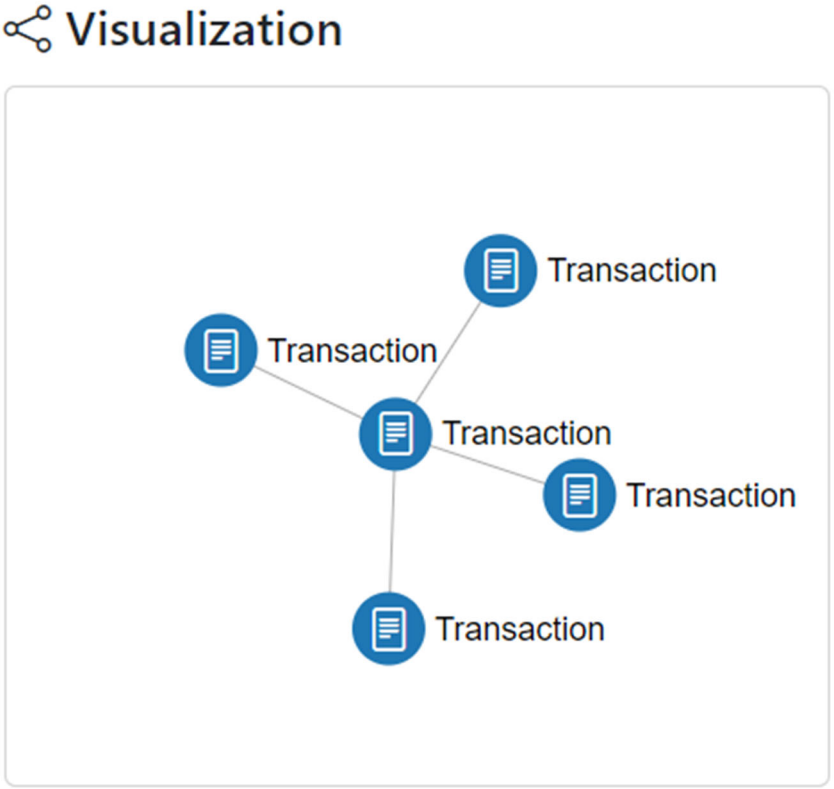

Рисунок 3.17 – Представлення результатів пошуку у вигляді графа

This is an invoice document. Here are the details mentioned in it:

- **Company Name and Address:**
  - East Repair Inc.
  - 1912 Harvest Lane
  - New York, NY 12210

0:00 / 2:37

Audio transcription: Can you please return this for me? Yeah, sure. We'll help you with that right after the meeting. Hang around a little bit, and we'll have somebody come and help. We'll get that returned, taken care of. My apologies that you had to use this unusual venue to accomplish what should be a much simpler task. We'll also look into the root cause of why that happened. But anybody else have anything they need to return? Jeff,

Рисунок 3.18 – Перетворення зображень та аудіо на текст

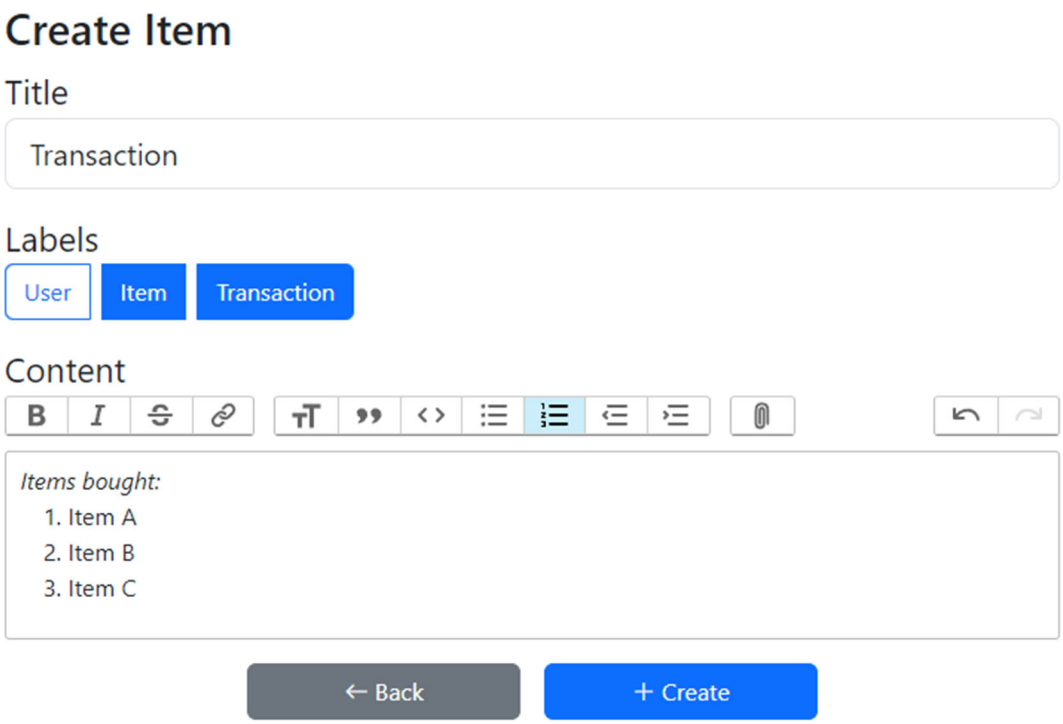

Рисунок 3.19 – Екран ручного створення нового вузла

Висновки до розділу 3

Отже, розроблена система підтримки бізнес-рішень є комплексним інструментом, що поєднує сучасні підходи до аналізу даних та ухвалення рішень на основі графових структур. Взаємодія із системою відбувається на програмному рівні за допомогою структурованого API або на візуальному рівні за допомогою інтуїтивного інтерфейсу. Архітектура системи забезпечує гнучкість та масштабованість застосунку, а графова база даних – ефективний пошук і обробку інформації. Натреновані нейронні мережі демонструють високі показники на різнотипних даних та інтегруються у систему для надання точних відповідей на питання користувачів. Використана велика мовна модель інтерпретує запити природною мовою з високою точністю і перетворює результат роботи нейронних мереж на практичні бізнес-ідеї.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ГРАФОВОЇ СИСТЕМИ ПІДТРИМКИ БІЗНЕС-РІШЕНЬ

4.1 Постановка задачі проєктування

Перед початком перетворення створеного прототипу на комерційний застосунок необхідно оцінити економічну доцільність реалізації розробленого програмного продукту, спрямованого на підтримку ухвалення бізнес-рішень. Для цього потрібно порівняти можливі стратегії розробки та подальшої підтримки програмного продукту з метою вибору оптимального підходу з економічного погляду. Одним із методів для виконання такого порівняння є функціонально-вартісний аналіз (ФВА).

Сутність ФВА полягає у комплексному техніко-економічному дослідженні функцій об’єкта [45]. Мета ФВА: мінімізація затрат об’єкта на стадіях проєктування, виробництва й експлуатації зі збереженням функціональності. Цілі ФВА: підвищення конкурентоспроможності продукту, зниження матеріаломісткості та трудомісткості продукції, підвищення економічності виробництва. ФВА дозволяє досягти кращого балансу між собівартістю і корисністю об’єкта шляхом пошуку оптимальних технічних рішень.

Після проведення аналізу потреб користувачів було визначено, що підсистеми мають задовольняти такі вимоги до програмного продукту:

- час на генерацію відповіді великою мовною моделлю не повинен перевищувати 2-3 секунди, час відповіді на запит до бази даних не повинен перевищувати 1 секунду;
- система повинна обробляти щонайменше 100 одночасних користувачів і 10 000 документів на годину без погіршення продуктивності;
- система повинна мати час безвідмовної роботи 99.9%;

- система повинна автоматично масштабуватися по горизонталі, щоб мати можливість впоратися з навантаженням, що зростає;
- частка помилок при перетворенні аудіо на текст (WER) не повинна перевищувати 10.0%, а при перетворенні зображень на текст – 15.0%;
- база знань повинна мати можливість постійного оновлення шляхом інтеграції нових інформаційних ресурсів;
- інтерфейс повинен бути інтуїтивно зрозумілим і зручним для користувача, з чіткою навігацією та адаптацією під розмір екрана;
- результати повинні відображатися за допомогою інтерактивного графа, що підтримує масштабування та переміщення;
- дані повинні бути зашифровані під час передачі з використанням TLS 1.2 або вище, а під час збереження – з використанням AES-256;
- інформація користувачів повинна бути конфіденційною;
- доступ до даних повинен бути контрольований на основі ролей користувачів.

4.2 Обґрунтування функцій програмного продукту

На основі поставлених вимог виділимо основні функції продукту та їхні можливі реалізації. Головна функція F_0 – розробка ПП, що дозволяє ухвалювати стратегічні бізнес-рішення шляхом аналізу гетерогенної інформації у базі даних і генерації рекомендацій за допомогою великої мовної моделі. Функція F_0 містить кілька складових, імплементацію яких треба ухвалити перед початком розробки комерційної версії продукту: F_1 – мова програмування; F_2 – бібліотека машинного навчання; F_3 – база даних; F_4 – велика мовна модель;

Кожна з основних функцій може мати декілька варіантів реалізації:

Функція F_1.

a) Python.

b) Java.

Функція F_2.

a) TensorFlow.

b) PyTorch .

Функція F_3.

a) MySQL.

b) Neo4j.

Функція F_4.

a) OpenAI GPT-4o (API).

b) Meta Llama (локальна).

Варіанти реалізації основних функцій наведені у морфологічній карті системи на рис. 4.1.

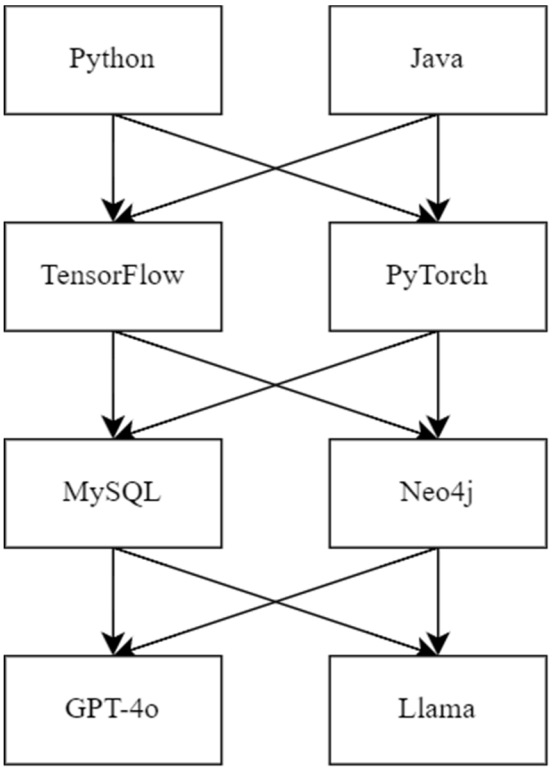

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця представлена в табл. 4.1.

Таблиця 4.1 – Позитивно-негативна матриця функцій продукту

Функція	*Варіант*	*Переваги*	*Недоліки*
F_1	a	Широке використання в машинному навчанні, велика кількість бібліотек, простота вивчення	Нижча продуктивність, складність паралелізації обчислень
	b	Висока продуктивність, надійна підтримка паралелізму	Складніший синтаксис, менш розвинені бібліотеки машинного навчання
F_2	a	Велика екосистема, більша кількість навчальних посібників	Поступове зменшення підтримки бібліотеки, менша стабільність
	b	Гнучкість використання, простота вивчення, популярність в індустрії та дослідженнях	Менша розвиненість і кількість навчальних матеріалів
F_3	a	Широке використання, простота вивчення	Складність обробки гетерогенних даних та графів знань
	b	Оптимізованість для графічних даних, ефективна обробка складних взаємозв’язків	Менша розвиненість, вища складність
F_4	a	Найбільша потужність, проста інтеграція	Залежність від зовнішнього сервісу, потенційні проблеми з конфіденційністю
	b	Відсутність залежності від зовнішнього сервісу, контроль над даними	Менша потужність, потреба у значних обчислювальних ресурсах

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функції варто відкинути, бо вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1.

Надаємо перевагу розвиненій екосистемі бібліотек машинного навчання і простоті роботи, тому обираємо варіант a і відкидаємо варіант b.

Функція F_2.

Особливу важливість становить гнучкість і активна розробка бібліотеки, тому обираємо варіант b і відкидаємо варіант a.

Функція F_3.

Система працює зі складними зв'язками між гетерогенними даними, тому обираємо варіант b і відкидаємо варіант a.

Функція F_4.

Обидва варіанти є прийнятними та будуть оцінені надалі.

Отже, будемо розглядати такі варіанти реалізації ПП:

- $F_1a - F_2b - F_3b - F_4a$
- $F_1a - F_2b - F_3b - F_4b$

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб схарактеризувати програмний продукт, будемо використовувати такі параметри (табл. 4.2):

- X_1 – швидкодія мови програмування;
- X_2 – простота написання коду для машинного навчання;
- X_3 – продуктивність бази даних;
- X_4 – якість відповіді моделі.

Таблиця 4.2 – Позитивно-негативна матриця функцій продукту

Параметр	*Умовне позначення*	*Одиниця виміру*	*Значення параметра*		
			граничне	*середнє*	*цільове*
Швидкодія мови програмування	X_1	$\frac{\text{оп}}{\text{с}}$	$5E4$	$2E5$	$8E5$
Простота написання коду для машинного навчання	X_2	рядки коду	1000	500	100
Продуктивність бази даних	X_3	$\frac{\text{запитів}}{\text{с}}$	$1E5$	$5E5$	$2E6$
Якість відповіді моделі	X_4	рейтинг Ело	800	1200	1300

За даними таблиці 4.2 будуються бальні оцінки характеристик параметрів, що наведені на рисунках 4.2 – 4.5.

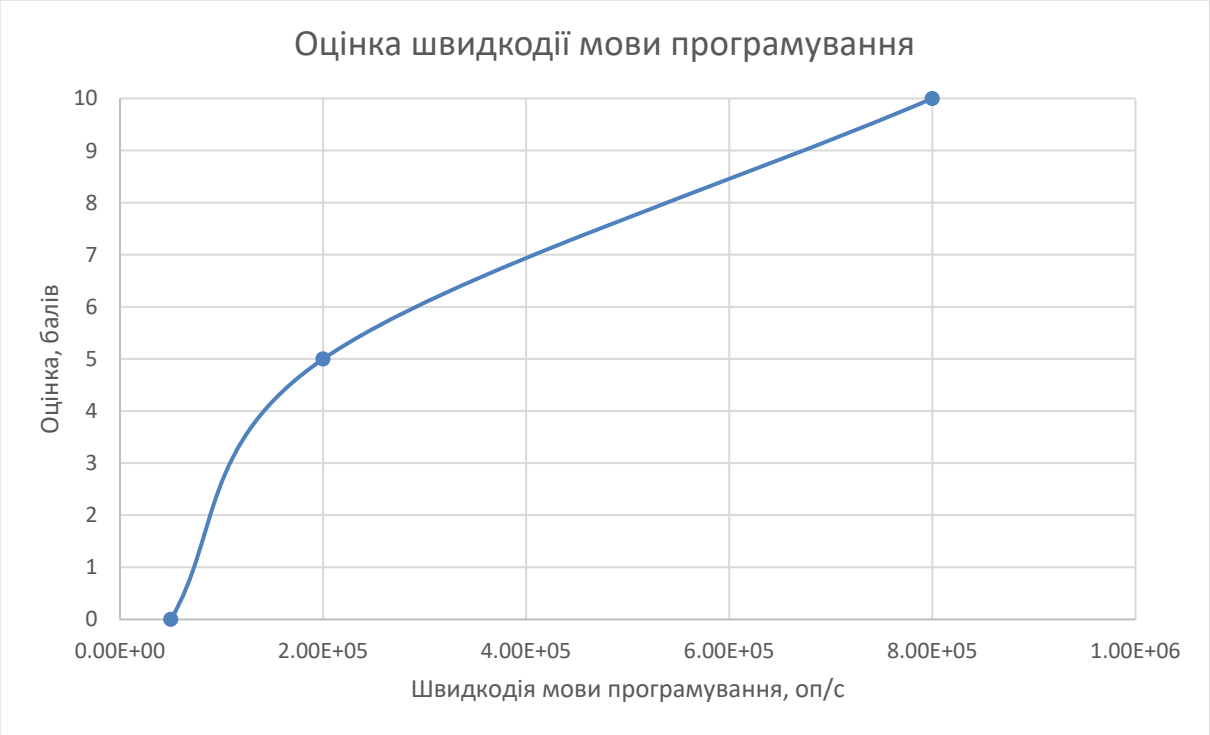

Рисунок 4.2 – Оцінка швидкодії мови програмування

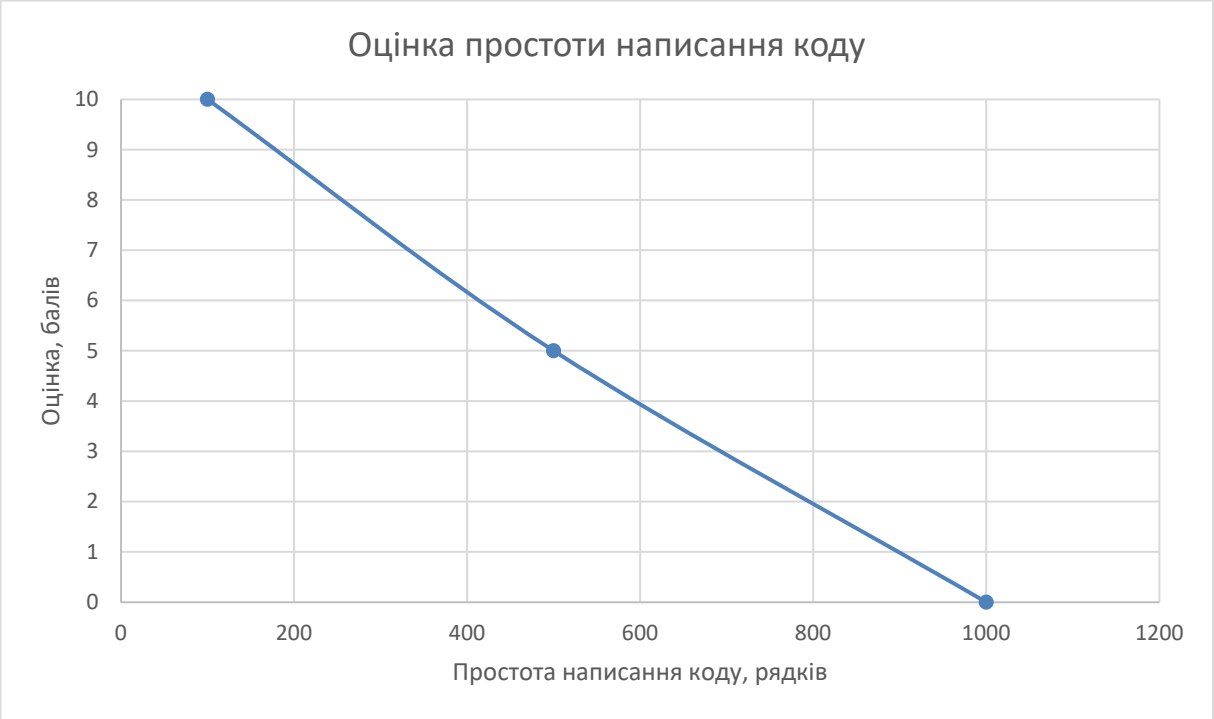

Рисунок 4.3 – Оцінка простоти написання коду для машинного навчання

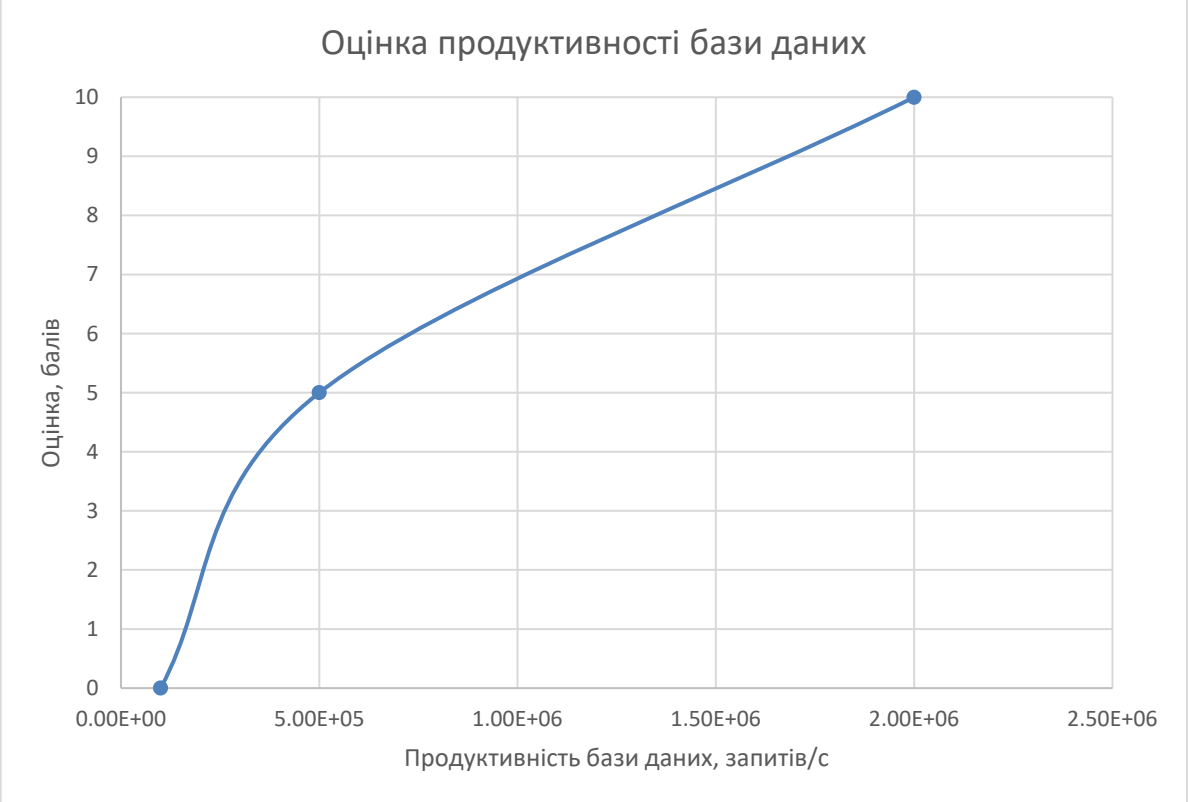

Рисунок 4.4 – Продуктивність бази даних

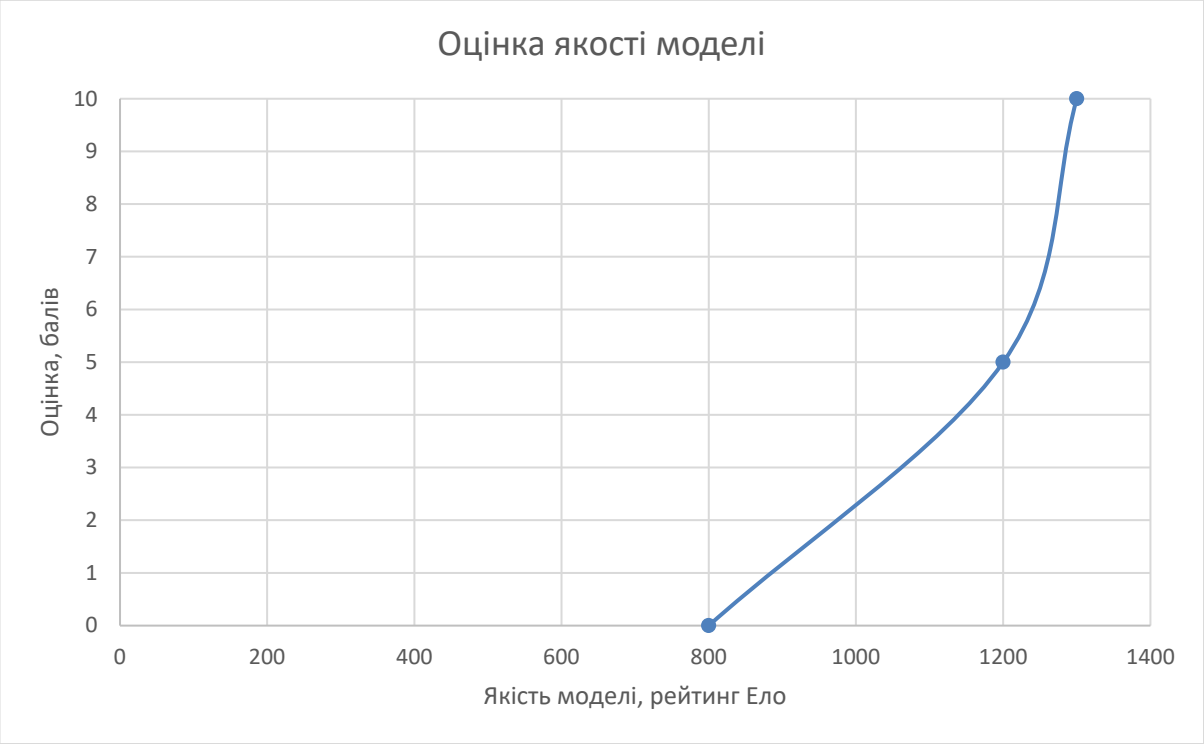

Рисунок 4.5 – Якість відповіді моделі

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний член експертної комісії оцінює ступінь важливості кожного параметра для конкретно поставленої цілі – розробки програмного продукту, що дає найкращі результати для підтримки ухвалення бізнес-рішень.

Значущість кожного параметра визначається методом попарного порівняння. Оцінку проводить комісія з 7 експертів. Визначення коефіцієнтів значущості передбачає:

– визначення рівня значущості параметра шляхом присвоєння різних рангів кожному параметру;

- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнта значущості.

Результати процесу експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позн.	*Назва*	*Од. виміру*	*Ранг за оцінкою експерта*							$\sum$	Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X_1	Швидк. мови прогр.	$\frac{\text{оп}}{\text{с}}$	4	3	3	3	4	4	4	25	7.5	56.25
X_2	Простота написання коду	рядки коду	3	4	4	4	2	3	3	23	5.5	30.25
X_3	Продукт. бази даних	$\frac{\text{запитів}}{\text{с}}$	2	1	2	2	3	2	1	13	−4.5	20.25
X_4	Якість відповіді моделі	рейтинг Ело	1	2	1	1	1	1	2	9	−8.5	72.25
	Разом		10	10	10	10	10	10	10	70	0	179.00

Для перевірки степеня достовірності експертних оцінок визначимо такі параметри:

- сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів,

n – кількість параметрів;

– середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17.5;$$

– відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_{\text{i}} = R_i - T,$$

причому сума відхилень за всіма параметрами повинна дорівнювати 0;

– загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_{\text{i}}^2 = 81.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 179}{7^2(4^3-4)} \approx 0.73 > W_k = 0.67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, що дорівнює 0.67.

Скориставшись результатами ранжування, проведемо попарне порівняння всіх параметрів і результати занесемо у табл. 4.4.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	*Експерти*							*Кінцева оцінка*	*Числове значення*
	1	*2*	*3*	*4*	*5*	*6*	*7*		
X_1 і X_2	$<$	$>$	$>$	$>$	$<$	$<$	$<$	$<$	0.5
X_1 і X_3	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	0.5
X_1 і X_4	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	0.5
X_2 і X_3	$<$	$<$	$<$	$<$	$>$	$<$	$<$	$<$	0.5
X_2 і X_4	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	0.5
X_3 і X_4	$<$	$>$	$<$	$<$	$<$	$<$	$>$	$<$	0.5

Числове значення a_{ij}, що визначає ступінь переваги i-го параметра над j-м, знаходять за формулою:

$$a_{ij} = \begin{cases} 1.5, X_i > X_j \\ 1.0, X_i = X_j. \\ 0.5, X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$. Для кожного параметра зробимо розрахунок вагомості $K_{\text{ві}}$ (табл. 4.5). Відносні оцінки розраховуються декілька разів, доки наступні значення не будуть незначно відрізнятися від попередніх. На другому і наступних кроках відносні оцінки розраховуються за допомогою b_i':

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i},$$

$$b_i = \sum_{i=1}^{N} a_{ij},$$

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'},$$

$$b_i' = \sum_{i=1}^{N} a_{ij}\, b_j.$$

Таблиця 4.5 – Розрахунок вагомості параметрів

X_i	X_j				*I ітер.*		*II ітер.*		*III ітер.*	
	X_1	X_2	X_3	X_4	$b_i^{(1)}$	$K_{\text{ві}}^{(1)}$	$b_i^{(2)}$	$K_{\text{ві}}^{(2)}$	$b_i^{(3)}$	$K_{\text{ві}}^{(3)}$
X_1	1.0	0.5	0.5	0.5	2.5	0.16	9.25	0.15	34.125	0.16
X_2	1.5	1.0	0.5	0.5	3.5	0.22	12.25	0.21	44.875	0.21
X_3	1.5	1.5	1.0	0.5	4.5	0.28	16.25	0.28	59.125	0.27
X_4	1.5	1.5	1.5	1.0	5.5	0.34	21.25	0.36	77.875	0.36
Разом:					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначимо рівень якості кожного варіанта виконання основних функцій (табл. 4.6). Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується за формулою:

$$K_{K,j} = \sum_{i=1}^{n} K_{\text{ві},j} B_{i,j},$$

де n – кількість параметрів;

$K_{\text{ві}}$ – коефіцієнт вагомості i-го параметра;

B_i – оцінка i-го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Функція	*Варіант*	*Параметр*	*Значення*	*Оцінка*	*Коефіцієнт вагомості*	*Коефіцієнт рівня якості*
F_1	a	X_1	$2E5$	6.0	0.16	0.96
F_2	b	X_2	200	8.5	0.21	1.785
F_3	b	X_3	$1E6$	7.0	0.27	1.89
F_4	a	X_4	1300	10.0	0.36	3.6
	b		1200	5.0		1.8

За даними з табл. 4.6 визначаємо рівень якості кожного з варіантів:

$$K_K = \sum_{j=1}^{n} K_{K,j},$$

$$K_{K_1} = 0.96 + 1.785 + 1.89 + 3.6 = 8.235,$$

$$K_{K_2} = 0.96 + 1.785 + 1.89 + 1.8 = 6.435.$$

Отже, кращим є перший варіант, $F_1a - F_2b - F_3b - F_4a$, для якого коефіцієнт технічного рівня має найбільше значення – 8.235, тобто реалізація програмного продукту мовою програмування Python з використанням бібліотеки машинного навчання PyTorch, бази даних Neo4j та великої мовної моделі OpenAI GPT-4o найкраще задовольняє технічні потреби, що були поставлені перед комерційним застосунком.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості. Обидва варіанти складаються з завдань 1–8.

1. Розробка конкретних бізнес-кейсів для розв’язання за допомогою графової інформаційної системи.

2. Збір різнотипних даних із загальнодоступних джерел, наприклад, із фінансових звітів, новин, маркетингових досліджень та соціальних мереж.
3. Очищення та попередня обробка інформації для забезпечення узгодженості та точності даних.
4. Інтеграція моделей image-to-text та audio-to-text для перетворення зображень та аудіо на текст.
5. Побудова графової бази даних, де вузли представляють сутності (компанії, продукти, особи), а ребра – зв’язки між ними.
6. Створення графової нейронної мережі та її навчання для класифікації вузлів та прогнозування зв’язків.
7. Візуалізація отриманих результатів у вигляді графа.
8. Оцінка результатів прогнозування та порівняння різних архітектур нейронних мереж.

Варіант 1 додатково містить завдання 9.

9. Інтеграція системи з великою мовною моделлю шляхом використання API.

Варіант 2 додатково містить завдання 10.

10. Розміщення великої мовної моделі на власному сервері, оптимізація пов’язаної інфраструктури, захист від зловмисних втручань.

Група складності алгоритму, ступінь новизни та вид використаної інформації для цих завдань занесено в табл. 4.7.

Таблиця 4.7 – Група складності алгоритму, ступінь новизни та вид використаної інформації для поставлених завдань

Завдання	*Група складності алгоритму*	*Ступінь новизни*	*Вид використаної інформації*
1	2	Б	ПІ
2	2	В	ПІ
3	3	Г	БД
4	2	Б	БД
5	2	В	БД
6	1	А	БД
7	3	Г	БД
8	1	А	НДІ
9	3	Г	БД
10	1	Б	БД

Проведемо розрахунок норм часу на виконання кожного завдання. Загальна трудомісткість обчислюється як:

$$T_i = T_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М},$$

де $T_Р$ – трудомісткість розробки ПП;

$К_П$ – коригувальний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність інформації;

$К_М$ – коефіцієнт мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Тоді загальна трудомісткість поставлених завдань:

$$T_1 = 27 \cdot 1.8 \cdot 0.8 = 38.88 \text{ людино-дня},$$

$$T_2 = 19 \cdot 1.2 \cdot 0.8 = 18.24 \text{ людино-дня},$$

$$T_3 = 8 \cdot 0.3 = 2.40 \text{ людино-дня},$$

$$T_4 = 27 \cdot 0.9 \cdot 0.8 \cdot 0.8 = 15.55 \text{ людино-дня},$$

$$\mathrm{Т}_5 = 19 \cdot 0.6 = 11.40 \text{ людино-дня},$$
$$\mathrm{Т}_6 = 90 \cdot 1.42 \cdot 0.8 = 102.24 \text{ людино-дня},$$
$$\mathrm{Т}_7 = 8 \cdot 0.3 = 2.40 \text{ людино-дня},$$
$$\mathrm{Т}_8 = 90 \cdot 1.42 \cdot 0.8 = 102.24 \text{ людино-дня},$$
$$\mathrm{Т}_9 = 8 \cdot 0.3 = 2.40 \text{ людино-дня},$$
$$\mathrm{Т}_{10} = 64 \cdot 1.01 = 64.64 \text{ людино-дня}.$$

Складемо трудомісткість відповідних завдань для обох обраних варіантів реалізації програми:

$$\mathrm{Т}_I = (38.88 + 18.24 + \cdots + 102.24 + 2.40) \cdot 8 = 2366 \text{ людино-годин},$$
$$\mathrm{Т}_{II} = (38.88 + 18.24 + \cdots + 102.24 + 64.64) \cdot 8 = 2863.92 \text{ людино-години}.$$

Отже, найвищу трудомісткість має варіант II.

У розробці беруть участь два програмісти з окладом 40 000 грн, один спеціаліст в області машинного навчання з окладом 60 000 грн. Визначимо середню зарплату за годину за формулою:

$$\mathrm{С_Ч} = \frac{1}{n}\sum_{i=1}^{n}\frac{\mathrm{М}_i}{T \cdot t},$$

де $\mathrm{М}_i$ – місячний оклад i-го працівника;

T – кількість робочих днів на місяць;

t – кількість робочих годин на день:

$$\mathrm{С_Ч} = \frac{40000+40000+60000}{3 \cdot 21 \cdot 8} = 277.78 \frac{\text{грн}}{\text{год}}.$$

Тоді розрахуємо заробітну плату за формулою:

$$\mathrm{С_{ЗП}} = \mathrm{С_Ч} \cdot T_i \cdot \mathrm{К_Д},$$

де $С_Ч$ – величина погодинної оплати праці програміста;

T_i – трудомісткість i-го завдання;

$К_Д$ – норматив, що враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$С_{ЗП_1} = 277.78 \cdot 2366 \cdot 1.2 = 788\,672.98 \text{ грн},$$
$$С_{ЗП_2} = 277.78 \cdot 2863.92 \cdot 1.2 = 954\,647.64 \text{ грн}.$$

Відрахування на єдиний соціальний внесок становить 22%:

$$С_{ВІД_1} = С_{ЗП_1} \cdot 0.22 = 173\,508.06 \text{ грн},$$
$$С_{ВІД_2} = С_{ЗП_2} \cdot 0.22 = 210\,022.48 \text{ грн}.$$

Визначимо витрати на оплату однієї машино-години $С_М$. Оскільки одна ЕОМ обслуговує одного програміста з окладом $40\,000$ грн з коефіцієнтом зайнятості 0.2, то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 40000 \cdot 0.2 = 96\,000 \text{ грн}.$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 96000 \cdot 1.2 = 115\,200 \text{ грн}.$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 115200 \cdot 0.22 = 25\,344 \text{ грн}.$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 32 000 грн:

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 32000 = 11\,200 \text{ грн},$$

де $К_{ТМ}$ – коефіцієнт, що враховує витрати на транспортування та монтаж приладу для користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт і профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot К_Р \cdot Ц_{ПР} = 1.4 \cdot 0.08 \cdot 32000 = 3584 \text{ грн},$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ЕОМ за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t \cdot К_В = 233 \cdot 8 \cdot 0.35 = 652.4 \text{ год},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$ і $Д_С$ – відповідно кількість вихідних і святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин на день, $К_В$ – коефіцієнт використання приладу.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_C \cdot К_З \cdot Ц_{ЕН} = 652.4 \cdot 0.2 \cdot 0.3 \cdot 5.22 = 204.33 \text{ грн},$$

де N_C – середня споживча потужність приладу;

$К_З$ – коефіцієнт зайнятості приладу;

$Ц_{ЕН}$ – тариф на електроенергію ($Ц_{ЕН} = 521.755 \frac{коп.}{кВт \cdot год}$ для малих непобутових споживачів, ПрАТ «ДТЕК Київські електромережі», 2 клас напруги, з ПДВ, станом на 14.05.2024).

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 32000 \cdot 0.67 = 21\,440 \text{ грн.}$$

Тоді річні експлуатаційні витрати:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н =$$
$$= 115200 + 25344 + 11200 + 3584 + 204.33 + 21440 = 176\,972.33 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М-Г} = \frac{С_{ЕКС}}{Т_{ЕФ}} = \frac{176972.33}{652.4} = 271.26 \frac{грн}{год}.$$

Оскільки в такому випадку всі роботи, які пов’язані з розробкою програмного продукту, ведуться на ЕОМ, то витрати на оплату машинного часу:

$$С_{М_i} = С_{М-Г} \cdot Т_i,$$
$$С_{М_I} = 271.26 \cdot 2366 = 641\,801.16 \text{ грн},$$
$$С_{М_{II}} = 271.26 \cdot 2863.92 = 776\,866.94 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$С_{Н_i} = С_{ЗП_i} \cdot 0.67,$$
$$С_{Н_I} = 788672.98 \cdot 0.67 = 528\,410.90 \text{ грн},$$
$$С_{Н_{II}} = 954647.64 \cdot 0.67 = 639\,613.92 \text{ грн}.$$

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП_i} = С_{ЗП_i} + С_{ВІД_i} + С_{М_i} + С_{Н_i},$$
$$С_{ПП_I} = 788672.98 + 173\,508.06 + 641\,801.16 + 528\,410.90 =$$
$$= 2\,132\,393.10 \text{ грн},$$
$$С_{ПП_{II}} = 954647.64 + 210\,022.48 + 776\,866.94 + 639\,613.92 =$$
$$= 2\,581\,150.98 \text{ грн}.$$

4.7 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_i} = К_{К_i} / С_{ПП_i},$$
$$К_{ТЕР_I} = \frac{8.235}{2132393.10} = 3.86 \cdot 10^{-6},$$
$$К_{ТЕР_{II}} = \frac{6.435}{2581150.98} = 2.49 \cdot 10^{-6}.$$

Отже, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР_I} = 3.86 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування: Python;
- бібліотека машинного навчання: PyTorch;

- база даних: Neo4j;
- велика мовна модель: OpenAI GPT-4o (API).

Даний варіант реалізації програмного комплексу дає команді розробників зручну мову програмування, гнучку бібліотеку машинного навчання, пристосовану до гетерогенної інформації базу даних та потужну велику мовну модель. Проаналізуємо також інші аспекти виконання поставлених завдань за допомогою обраних інструментів.

1. Використання великої мовної моделі GPT-4o дозволяє забезпечити швидку генерацію відповіді (в межах 2-3 секунд) шляхом використання потужної обчислювальної інфраструктури OpenAI. Обрання бази даних Neo4j, що оптимізована для обробки графових даних, дозволяє зменшити час відповіді на запит до 1 секунди.
2. Python і PyTorch забезпечують високий рівень паралелізму і гнучкість у реалізації обчислювально-інтенсивних задач; Neo4j здатна ефективно обробляти великий обсяг даних та зв’язків між ними; використання GPT-4o додає можливість масштабування на стороні сервісу. Це дозволяє обробляти запити від щонайменше 100 одночасних користувачів і 10 000 документів на годину без погіршення продуктивності.
3. Використання інструментів PyTorch для побудови та навчання моделей машинного навчання дозволяє досягти високої точності перетворення аудіо та зображень на текст. Дотримання вимог до точності розпізнавання може бути забезпечене шляхом використання новітніх моделей та алгоритмів.
4. Розробка інтерфейсу з використанням Bootstrap та шаблонів Flask дозволяє створити зручний та адаптивний інтерфейс для різних пристроїв.

Висновки до розділу 4

Отже, апарат функціонально-вартісного аналізу дозволив оцінити економічну доцільність реалізації графової системи підтримки бізнес-рішень в індустрії. Було розглянуто основні функції та можливі варіанти їх реалізації, проведено порівняльний аналіз та розрахунок техніко-економічного рівня. Обрано найкращі варіанти реалізації функцій застосунку для дотримання поставлених вимог.

Під час аналізу було встановлено, що найбільш ефективною сукупністю компонентів системи є мова програмування Python, бібліотека машинного навчання PyTorch, база даних Neo4j та велика мовна модель OpenAI GPT-4o. Такий підхід забезпечує високу ефективність та конкурентоспроможність системи, що дозволяє ухвалювати стратегічні бізнес-рішення на основі аналізу гетерогенної інформації та генерації рекомендацій за допомогою великої мовної моделі.

ВИСНОВКИ

У цій роботі представлено процес комплексного дослідження та розробки інформаційно-аналітичної системи на основі графів, призначеної для підтримки ухвалення бізнес-рішень. Система використовує елементи теорії графів, методи машинного навчання та сучасні практики програмної інженерії для створення надійного, ефективного та універсального інструменту для бізнесу.

У першому розділі було досліджено фундаментальні принципи систем графового типу в бізнес-аналітиці. Розглянуто базові положення теорії графів та рівні аналізу графів у бізнес-системах, надано приклади використання графових систем для вирішення бізнес-завдань, наприклад для управління клієнтами, виявлення шахрайства та оптимізації логістики. Визначено постановку задачі, розв’язання якої стоїть перед розробленою системою.

У другому розділі було розглянуто застосування методів машинного навчання для аналізу графових структур у бізнесі. Зображено алгоритмічні підходи для побудови векторних представлень графових бізнес-даних та підкреслено їхню здатність перетворювати складні дані на придатний для моделей машинного навчання формат. Представлено бізнес-орієнтовані архітектури GNN, наприклад, GCN і GAT, що пристосовані до ефективного вирішення конкретних проблем у бізнесі. Розглянуто стратегії навчання GNN для завдань класифікації, кластеризації та регресії. Надано приклади представлення гетерогенних бізнес-даних у вигляді графа знань з подальшим доповненням зв’язків та виконанням прогностичних запитів.

У третьому розділі було описано розробку графової системи підтримки бізнес-рішень. Представлено архітектуру системи, що складається з графової бази даних, графової нейронної мережі та великої мовної моделі, і забезпечує масштабованість та надійність застосунку. Продемонстровано збір різнотипної інформації з класичних наборів даних, їхню попередню обробку, завантаження у графову базу даних та передачу інформації у графову нейронну мережу.

Проведено підбір гіперпараметрів мережі та оцінено ефективність шарів GCNConv, GATConv та TransformerConv у порівнянні з шарами лінійних перетворень. Візуалізовано результати роботи мереж та проаналізовано переваги графової парадигми для прогнозування комплексних даних із великою кількістю зв'язків. Пояснено структуру кінцевих точок для взаємодії з розробленою системою через API, що забезпечує безперешкодну інтеграцію з іншими бізнес-додатками та системами. Зображено інтерфейс користувача та механізм генерації практичних ідей шляхом уведення запиту в пошуковий рядок, його інтерпретації за допомогою великої мовної моделі, активації доцільних функцій нейронної мережі та отримання точної відповіді й візуалізації на основі даних, якими володіє бізнес.

У четвертому розділі було проведено функціонально-вартісний аналіз розробленої системи для оцінювання економічної доцільності комерційної реалізації створеного прототипу. Надано обґрунтування функціональних можливостей системи та параметрів програмного продукту шляхом визначення ключових показників ефективності системи. Обрано найбільш доцільний варіант реалізації системи за коефіцієнтом техніко-економічного рівня.

Для інтеграції розробленої системи в бізнес-процеси компаній рекомендується виконати такі кроки: визначити дані, що можуть бути ефективно представлені у вигляді графа, наприклад, клієнти, транзакції та продукти; завантажити дані за допомогою API або інтерфейсу застосунку; запустити процес навчання вбудованих нейронних мереж для їхньої адаптації до таких даних; увести потрібний запит у пошуковий рядок системи для отримання візуалізації та практичних ідей із наявної інформації.

Подальші дослідження можуть бути спрямовані на розробку більш ефективних графових алгоритмів, використання передових моделей ШІ, оптимізацію продуктивності системи на комплексних графах за великих навантажень, адаптацію системи до сфер фінансів та охорони здоров'я шляхом інтеграції специфічних для цих галузей знань, підтримку етичних практик та конфіденційності даних, інтеграцію системи з пристроями інтернету речей.

ПЕРЕЛІК ПОСИЛАНЬ

1. Tomaž Bratanic, Graph Algorithms for Data Science. Simon and Schuster, 2024.
2. Y. Ma and J. Tang, Deep learning on graphs. Cambridge, United Kingdom: Cambridge University Press, 2021.
3. A. Negro, Graph-Powered Machine Learning. Simon and Schuster, 2021.
4. W. L. William, Graph Representation Learning. Springer Nature, 2022.
5. M. Granovetter, “The Strength of Weak Ties: A Network Theory Revisited,” Sociological Theory, vol. 1, no. 6, pp. 201–233, 1983, URL: https://doi.org/10.2307/202051 (дата звернення: 14.05.2024).
6. V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” Journal of Statistical Mechanics: Theory and Experiment, vol. 2008, no. 10, p. P10008, Oct. 2008, URL: https://doi.org/10.1088/1742-5468/2008/10/p10008 (дата звернення: 14.05.2024).
7. J. Webber, “The #1 Platform for Connected Data. The Top 10 Use Cases of Graph Database Technology Unlock New Possibilities with Connected Data.” URL: https://go.neo4j.com/rs/710-RRC-335/images/Neo4j-Top-10-Use-Cases-EN-US.pdf (дата звернення: 14.05.2024).
8. B. Perozzi, R. Al-Rfou, and S. Skiena, “DeepWalk: Online Learning of Social Representations,” Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’14, pp. 701–710, 2014, URL: https://doi.org/10.1145/2623330.2623732 (дата звернення: 14.05.2024).
9. T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” arXiv.org, Sep. 07, 2013. URL: https://arxiv.org/abs/1301.3781 (дата звернення: 14.05.2024).

10. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” arXiv.org, 2013. URL: https://arxiv.org/abs/1310.4546 (дата звернення: 14.05.2024).
11. A. Grover and J. Leskovec, “node2vec: Scalable Feature Learning for Networks,” Jul. 2016, URL: https://doi.org/10.48550/arxiv.1607.00653 (дата звернення: 14.05.2024).
12. D. Duvenaud et al., “Convolutional Networks on Graphs for Learning Molecular Fingerprints,” arXiv.org, Nov. 03, 2015. URL: https://arxiv.org/abs/1509.09292 (дата звернення: 14.05.2024).
13. Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated Graph Sequence Neural Networks,” arXiv:1511.05493 [cs, stat], Sep. 2017, URL: https://arxiv.org/abs/1511.05493 (дата звернення: 14.05.2024).
14. S. Brin and L. Page, “The anatomy of a large-scale hypertextual Web search engine,” Computer Networks and ISDN Systems, vol. 30, no. 1–7, pp. 107–117, Apr. 1998, URL: https://doi.org/10.1016/s0169-7552(98)00110-x (дата звернення: 14.05.2024).
15. D. Easley and J. Kleinberg, Networks, crowds, and markets: reasoning about a highly connected world. New York: Cambridge University Press, 2010.
16. Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*(6), 386–408. URL: https://doi.org/10.1037/h0042519 (дата звернення: 14.05.2024).
17. G. Cybenko, “Approximation by superpositions of a sigmoidal function,” Mathematics of Control, Signals, and Systems, vol. 2, no. 4, pp. 303–314, Dec. 1989, URL: https://doi.org/10.1007/bf02551274 (дата звернення: 14.05.2024).
18. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” Nature, vol. 323, no. 6088, pp. 533–536, Oct. 1986, URL: https://doi.org/10.1038/323533a0 (дата звернення: 14.05.2024).
19. F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini, “The Graph Neural Network Model,” IEEE Transactions on Neural Networks,

vol. 20, no. 1, pp. 61–80, Jan. 2009, URL: https://doi.org/10.1109/tnn.2008.2005605 (дата звернення: 14.05.2024).

20. B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. Wiltschko, “A Gentle Introduction to Graph Neural Networks,” Distill, vol. 6, no. 8, Aug. 2021, URL: https://doi.org/10.23915/distill.00033 (дата звернення: 14.05.2024).
21. T. N. Kipf and M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks,” arXiv:1609.02907 [cs, stat], Feb. 2017, URL: https://arxiv.org/abs/1609.02907 (дата звернення: 14.05.2024).
22. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998, URL: https://doi.org/10.1109/5.726791 (дата звернення: 14.05.2024).
23. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph Attention Networks,” arXiv:1710.10903 [cs, stat], Feb. 2018, URL: https://arxiv.org/abs/1710.10903 (дата звернення: 14.05.2024).
24. S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” arXiv.org, 2015. URL: https://arxiv.org/abs/1502.03167 (дата звернення: 14.05.2024).
25. I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, Massachusetts: The Mit Press, 2016. URL: http://imlab.postech.ac.kr/dkim/class/csed514_2019s/DeepLearningBook.pdf (дата звернення: 14.05.2024).
26. N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” Journal of Machine Learning Research, vol. 15, pp. 1929–1958, 2014, URL: https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf (дата звернення: 14.05.2024).
27. J. Leskovec and H. Ren, "CS224W: Machine Learning with Graphs," Stanford University, [Online]. URL: http://cs224w.stanford.edu. (дата звернення: 14.05.2024).

28. W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive Representation Learning on Large Graphs,” arXiv:1706.02216 [cs, stat], Sep. 2018, URL: https://arxiv.org/abs/1706.02216 (дата звернення: 14.05.2024).
29. W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, “Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks,” Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 257–266, Jul. 2019, URL: https://doi.org/10.1145/3292500.3330925 (дата звернення: 14.05.2024).
30. F. Wu, T. Zhang, A. H. de Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger, “Simplifying Graph Convolutional Networks,” arXiv.org, Jun. 20, 2019. URL: https://arxiv.org/abs/1902.07153 (дата звернення: 14.05.2024).
31. R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, “Hierarchical Graph Representation Learning with Differentiable Pooling,” arXiv:1806.08804 [cs, stat], Feb. 2019, URL: https://arxiv.org/abs/1806.08804 (дата звернення: 14.05.2024).
32. M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, “Modeling Relational Data with Graph Convolutional Networks,” arXiv:1703.06103 [cs, stat], Oct. 2017, URL: https://arxiv.org/abs/1703.06103 (дата звернення: 14.05.2024).
33. N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, “Industry-scale Knowledge Graphs: Lessons and Challenges,” Queue, vol. 17, no. 2, pp. 48–75, Apr. 2019, URL: https://doi.org/10.1145/3329781.3332266 (дата звернення: 14.05.2024).
34. “Graph API - Documentation,” Facebook for Developers. URL: https://developers.facebook.com/docs/graph-api/ (дата звернення: 14.05.2024).
35. F. Suchanek, M. Alam, T. Bonald, L. Chen, P.-H. Paris, and J. Soria, “YAGO 4.5: A Large and Clean Knowledge Base with a Rich Taxonomy,” arXiv.org, Apr. 10, 2024. URL: https://arxiv.org/abs/2308.11884 (дата звернення: 14.05.2024).

36. A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, “Translating embeddings for modeling multirelational data,” in NIPS, 2013, pp. 2787–2795.
37. Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, “Learning Entity and Relation Embeddings for Knowledge Graph Completion,” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29, no. 1, Feb. 2015, URL: https://doi.org/10.1609/aaai.v29i1.9491 (дата звернення: 14.05.2024).
38. B. Yang, W. Yih, X. He, J. Gao, and L. Deng, “Embedding Entities and Relations for Learning and Inference in Knowledge Bases,” arXiv (Cornell University), May 2015, URL: https://doi.org/10.48550/arxiv.1412.6575 (дата звернення: 14.05.2024).
39. T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, “Complex Embeddings for Simple Link Prediction,” arXiv.org, Jun. 20, 2016. URL: https://arxiv.org/abs/1606.06357 (дата звернення: 14.05.2024).
40. K. Guu, J. Miller, and P. Liang, “Traversing Knowledge Graphs in Vector Space,” arXiv.org, Aug. 19, 2015. URL: https://arxiv.org/abs/1506.01094 (дата звернення: 14.05.2024).
41. H. Ren, W. Hu, and J. Leskovec, “Query2box: Reasoning over Knowledge Graphs in Vector Space using Box Embeddings,” arXiv (Cornell University), Jan. 2020, URL: https://doi.org/10.48550/arxiv.2002.05969 (дата звернення: 14.05.2024).
42. W. W. Zachary, “An Information Flow Model for Conflict and Fission in Small Groups,” Journal of Anthropological Research, vol. 33, no. 4, pp. 452–473, Dec. 1977, URL: https://doi.org/10.1086/jar.33.4.3629752 (дата звернення: 14.05.2024).
43. Z. Yang, W. W. Cohen, and R. Salakhutdinov, “Revisiting Semi-Supervised Learning with Graph Embeddings,” arXiv.org, May 26, 2016. URL: https://arxiv.org/abs/1603.08861 (дата звернення: 14.05.2024).

44. O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, “Pitfalls of Graph Neural Network Evaluation,” arXiv.org, Jun. 18, 2019. URL: https://arxiv.org/abs/1811.05868 (дата звернення: 14.05.2024).
45. З. Б. Литвин, "Функціонально-вартісний аналіз," Економічна думка, 2007.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Файлова ієрархія ІСППР має такий вигляд:

```
.venv
data/
    ├── cache/
    ├── transaction/
        ├── EllipticBitcoin/
        ├── transaction_edges.jsonl
        ├── transaction_nodes.jsonl
        ├── transactions_raw.pkl
        ├── transactions.pkl
db/
    ├── neo4j_connection.py
    ├── queries.py
models/
    ├── data_to_neo4j.ipynb
    ├── gcn.py
    ├── helpers.py
    ├── neo4j_to_ml.ipynb
    ├── transaction_classifier.pth
    ├── transaction.py
openai_tools/
    ├── transaction.py
routes/
    ├── auth.py
    ├── files.py
    ├── graph.py
    ├── main.py
    ├── openai_tools.py
static/
    ├── files/
    ├── icons/
    ├── scripts/
        ├── base.js
        ├── create_item.js
        ├── get_item.js
        ├── search.js
        ├── trix.js
    ├── styles/
        ├── styles.css
templates/
    ├── base.html
    ├── create_item.html
    ├── date_badges.html
    ├── get_item.html
    ├── graph.html
    ├── index.html
```

```
    ├── label_badges.html
    ├── search.html
    ├── searchbar.html
    ├── trix.html
utils/
    ├── helpers.py
.env
.gitignore
app.py
config.py
README.md
requirements.txt
test_app.py
```

Вміст файлів .py та .js представлений нижче.

db/neo4j_connection.py

```
from neo4j import GraphDatabase
from config import Config

class Neo4jConnection:
    def __init__(self, uri, auth):
        self.__uri = uri
        self.__auth = auth
        self.__driver = None
        try:
            self.__driver = GraphDatabase.driver(self.__uri, auth=(self.__auth))
            # self.create_indexes()
        except Exception as e:
            print("Failed to create the driver:", e)

    def close(self):
        if self.__driver is not None:
            self.__driver.close()

    def query(self, query, parameters=None, db=None):
        assert self.__driver is not None, "Driver not initialized!"
        session = None
        response = None
        try:
            session = self.__driver.session(database=db) if db is not None else self.__driver.session()
            response = list(session.run(query, parameters))
        except Exception as e:
            print("Query failed:", e)
        finally:
            if session is not None:
                session.close()
        return response

    def create_indexes(self):
        index_queries = [
```

```
            f"""
            CREATE VECTOR INDEX `item_embedding_index` IF NOT EXISTS
            FOR (n:Item)
            ON (n.embedding)
            OPTIONS {{indexConfig: {{
                `vector.dimensions`: {Config.OPENAI_EMBEDDING_DIMENSIONS},
                `vector.similarity_function`: 'cosine'
            }}}}
            """,
            """
            CREATE CONSTRAINT item_id_constraint IF NOT EXISTS
            FOR (n:Item)
            REQUIRE n.id IS UNIQUE
            """,
            """
            CREATE INDEX item_title_index IF NOT EXISTS
            FOR (n:Item)
            ON (n.title)
            """,
            """
            CREATE INDEX item_created_at_index IF NOT EXISTS
            FOR (n:Item)
            ON (n.created_at)
            """
        ]

        for query in index_queries:
            try:
                self.query(query)
                print(f"Index created successfully or already exists for query: {query.strip().split()[2]}")
            except Exception as e:
                print(f"Failed to create index for query: {query.strip().split()[2]}, error: {e}")

conn = Neo4jConnection(
    uri=Config.NEO4J_URI,
    auth=(Config.NEO4J_ADMIN_USERNAME, Config.NEO4J_ADMIN_PASSWORD)
)
```

db/queries.py

```
from db.neo4j_connection import conn
from utils.helpers import generate_id, create_openai_embedding, create_item_embedding
from datetime import datetime, timezone
from config import Config


def get_node(id):
    query = """
        MATCH (n) WHERE n.id = $id
        RETURN properties(n) AS node, labels(n) AS labels
    """
    records = conn.query(
```

```
        query=query,
        parameters={'id': id},
        db="neo4j"
    )
    if records:
        node = records[0]['node']
        node['labels'] = records[0]['labels']
        return node
    else:
        return None


def list_nodes(filters=None, query="", page=1, page_size=10, sort_by=None, sort_order=None,
start_date=None, end_date=None):
    skip = (page - 1) * page_size
    labels = filters.pop('labels', []) if filters else []
    where_clause = ""
    filter_conditions = []

    if filters:
        for key, value in filters.items():
            filter_conditions.append(f"n.{key} = ${key}")

    if labels:
        label_conditions = [f"'{label}' IN labels(n)" for label in labels]
        filter_conditions.extend(label_conditions)

    if start_date:
        filter_conditions.append("n.created_at >= $start_date")

    if end_date:
        filter_conditions.append("n.created_at <= $end_date")

    if sort_by is None:
        sort_by = 'created_at'

    if sort_order is None:
        sort_order = 'DESC'

    if filter_conditions:
        where_clause = "WHERE " + " AND ".join(filter_conditions)

    if query:
        neo4j_query = f"""
            CALL db.index.vector.queryNodes('item_embedding_index', {page * page_size},
$embedding)
            YIELD node AS n, score
            {where_clause}

            WITH n, score
            ORDER BY score DESC, n.{sort_by} {sort_order}, n.id ASC
```

```
            SKIP {skip} LIMIT {page_size}

            RETURN properties(n) AS properties, labels(n) AS labels, score
        """
    else:
        neo4j_query = f"""
            MATCH (n)
            {where_clause}

            WITH n
            ORDER BY n.{sort_by} {sort_order}, n.id ASC

            SKIP {skip} LIMIT {page_size}

            RETURN properties(n) AS properties, labels(n) AS labels
        """

    parameters = filters.copy() if filters else {}
    parameters.update({
        'embedding': create_openai_embedding(query),
        'start_date': start_date,
        'end_date': end_date
    })

    records = conn.query(
        query=neo4j_query,
        parameters=parameters,
        db="neo4j"
    )
    results = []
    for record in records:
        node = record['properties']
        node['labels'] = record['labels']
        node['score'] = record.get('score', None)
        node.pop('embedding', None)
        results.append(node)

    return results


def create_node(properties):
    labels = properties.pop('labels', [])
    labels.append('Item')

    slug = properties.pop('slug', None)
    title = properties.pop('title', None)
    content = properties.pop('content', None)

    properties['id'] = generate_id(labels=labels, slug=slug)

    properties['title'] = title
    properties['content'] = content
```

```
    time_now = datetime.now(timezone.utc).strftime('%Y-%m-%dT%H:%M:%SZ')
    properties['created_at'] = time_now
    properties['updated_at'] = time_now
    properties['embedding'] = create_item_embedding({
        'title': title,
        'labels': labels,
        'content': content
    })

    query = f"""
        CREATE (n:{':'.join(labels)} $properties)
        RETURN n.id AS id
    """

    records = conn.query(
        query=query,
        parameters={'properties': properties},
        db="neo4j"
    )

    if records:
        return {
            'id': records[0]['id']
        }
    else:
        return None


def update_node(id, properties):
    time_now = datetime.now(timezone.utc).strftime('%Y-%m-%dT%H:%M:%SZ')

    labels = properties.pop('labels', [])
    label_clause = f", n:{':'.join(labels)}" if labels else ""
    title = properties.pop('title', None)
    content = properties.pop('content', None)

    properties['title'] = title
    properties['content'] = content

    properties['updated_at'] = time_now
    properties['embedding'] = create_item_embedding({
        'title': title,
        'labels': labels,
        'content': content
    })

    query = f"""
        MATCH (n) WHERE n.id = $id
        SET n += $properties{label_clause}
        RETURN n.id AS id
    """
```

```
    records = conn.query(
        query=query,
        parameters={'id': id, 'properties': properties},
        db="neo4j"
    )

    if records:
        return {
            'id': records[0]['id']
        }
    else:
        return None


def delete_node(id):
    query = """
        MATCH (n) WHERE n.id = $id
        DETACH DELETE n
        RETURN n
    """
    result = conn.query(
        query=query,
        parameters={'id': id},
        db="neo4j"
    )
    return result if result else None


def create_node_batch(nodes):
    time_now = datetime.now(timezone.utc).strftime('%Y-%m-%dT%H:%M:%SZ')

    for node in nodes:
        slug = node.pop('slug', None)
        title = node.pop('title', None)
        content = node.pop('content', None)

        node['labels'].append('Item')

        embedding = create_item_embedding({
            'title': title,
            'labels': node['labels'],
            'content': content
        })

        node.update({
            'id': generate_id(labels=node['labels'], slug=slug),
            'title': title,
            'content': content,
            'created_at': time_now,
            'updated_at': time_now,
            'embedding': embedding
        })
```

```
    query = f"""
        UNWIND $batch AS node
        CREATE (n)
        SET n = node.properties
        WITH n, node
        CALL apoc.create.addLabels(n, node.labels)
        YIELD node AS updatedNode
        RETURN updatedNode.id AS id
    """

    parameters = {'batch': [{'properties': node, 'labels': node.pop('labels')} for node in nodes]}

    records = conn.query(
        query=query,
        parameters=parameters,
        db="neo4j"
    )

    return [{'id': record['id']} for record in records]


def get_edge(id):
    query = """
        MATCH (src)-[r]->(dst)
        WHERE r.id = $id
        RETURN properties(r) AS properties, type(r) AS type, src.id AS src, dst.id AS dst
    """
    records = conn.query(
        query=query,
        parameters={'id': id},
        db="neo4j"
    )
    if records:
        edge = records[0]['properties']
        edge.update({
            'src': records[0]['src'],
            'dst': records[0]['dst'],
            'type': records[0]['type']
        })
        return edge


def list_edges(filters=None, page=1, page_size=10, sort_by=None, sort_order=None,
start_date=None, end_date=None):
    skip = (page - 1) * page_size
    relationship_type = filters.pop('type', None) if filters else None
    where_clause = ""
    filter_conditions = []

    if filters:
        for key, value in filters.items():
            filter_conditions.append(f"r.{key} = ${key}")
```

```
if relationship_type:
    filter_conditions.extend([f"type(r) = '{relationship_type}'"])

if start_date:
    filter_conditions.append("r.created_at >= $start_date")

if end_date:
    filter_conditions.append("r.created_at <= $end_date")

if filter_conditions:
    where_clause = "WHERE " + " AND ".join(filter_conditions)

if sort_by is None:
    sort_by = 'created_at'

if sort_order is None:
    sort_order = 'DESC'

neo4j_query = f"""
    MATCH (src)-[r]->(dst)
    {where_clause}

    WITH r, src, dst
    ORDER BY r.{sort_by} {sort_order}, r.id ASC

    SKIP {skip} LIMIT {page_size}

    RETURN properties(r) AS properties, type(r) AS type, src.id AS src, dst.id AS dst
"""

parameters = filters.copy() if filters else {}
parameters.update({
    'start_date': start_date,
    'end_date': end_date
})

records = conn.query(
    query=neo4j_query,
    parameters=parameters,
    db="neo4j"
)
results = []
for record in records:
    edge = record['properties']
    edge.update({
        'src': record['src'],
        'dst': record['dst'],
        'type': record['type']
    })

    results.append(edge)
```

```
    return results


def create_edge(properties):
    time_now = datetime.now(timezone.utc).strftime('%Y-%m-%dT%H:%M:%SZ')

    src_id = properties.pop('src', None)
    dst_id = properties.pop('dst', None)
    edge_type = properties.pop('type', 'RELATED')
    slug = properties.pop('slug', None)

    properties['id'] = generate_id(labels=[edge_type], slug=slug)
    properties['created_at'] = time_now
    properties['updated_at'] = time_now

    query = f"""
        MATCH (src), (dst)
        WHERE src.id = $src_id AND dst.id = $dst_id
        CREATE (src)-[r:{edge_type} $properties]->(dst)
        RETURN r.id AS id
    """
    records = conn.query(
        query=query,
        parameters={'src_id': src_id, 'dst_id': dst_id, 'properties': properties},
        db="neo4j"
    )
    if records:
        return {
            'id': records[0]['id']
        }
    else:
        return None


def create_edge_batch(edges):
    time_now = datetime.now(timezone.utc).strftime('%Y-%m-%dT%H:%M:%SZ')

    for edge in edges:
        edge_type = edge.get('type', 'RELATED')
        slug = edge.pop('slug', None)

        edge.update({
            'id': generate_id(labels=[edge_type], slug=slug),
            'created_at': time_now,
            'updated_at': time_now
        })

    query = """
        UNWIND $batch AS edge
        MATCH (src:Item {id: edge.src}), (dst:Item {id: edge.dst})
        CALL apoc.create.relationship(src, edge.type, edge.properties, dst) YIELD rel
```

```
        RETURN rel.id AS id
    """

    parameters = {'batch': [{'src': edge['src'], 'dst': edge['dst'], 'properties': edge, 'type':
edge.pop('type', 'RELATED')} for edge in edges]}

    records = conn.query(
        query=query,
        parameters=parameters,
        db="neo4j"
    )

    return [{'id': record['id']} for record in records]


def list_node_labels():
    query = """
        CALL db.labels()
    """
    records = conn.query(
        query=query,
        db="neo4j"
    )
    return [record['label'] for record in records]


def get_graph_neighborhood(ids, node_limit=10, edge_limit=10):
    query = f"""
        WITH {ids} AS nodeIds
        MATCH (n)
        WHERE n.id IN nodeIds
        WITH n
        LIMIT {node_limit}
        OPTIONAL MATCH (n)-[r]-(m)
        WITH n, r, m
        LIMIT {edge_limit}
        WITH COLLECT(DISTINCT n) AS nodes, COLLECT(DISTINCT m) AS neighbors,
COLLECT(DISTINCT r) AS relationships
        WITH apoc.coll.toSet(nodes + neighbors) AS uniqueNodes, relationships
        RETURN {{
            nodes: [node IN uniqueNodes WHERE node IS NOT NULL | {{
                id: node.id,
                title: node.title,
                labels: labels(node)
            }}],
            edges: [rel IN relationships WHERE rel IS NOT NULL | {{
                src: startNode(rel).id,
                dst: endNode(rel).id
            }}]
        }} AS graph
    """
    records = conn.query(
```

```
        query=query,
        db="neo4j"
    )
    return records[0]['graph'] if records else None


def list_indexes():
    query = """
        SHOW INDEXES
    """
    records = conn.query(
        query=query,
        db="neo4j"
    )
    return records


def create_index(label, property='embedding'):
    query = f"""
        CREATE VECTOR INDEX `{label}_embeddings` IF NOT EXISTS
        FOR (n:{label})
        ON (n.{property})
        OPTIONS {{indexConfig: {{
            `vector.dimensions`: { Config.OPENAI_EMBEDDING_DIMENSIONS },
            `vector.similarity_function`: 'cosine'
        }}}}
    """
    records = conn.query(
        query=query,
        db="neo4j"
    )
    return records
```

models/data_to_neo4j.ipynb

```
import os

if os.path.basename(os.getcwd()) != 'graph':
    os.chdir('..')

from config import Config

data_dir = 'data/transaction'
base_url = Config.BASE_URL

node_url = f"{base_url}/api/nodes/batch"
edge_url = f"{base_url}/api/edges/batch"

node_file = f'{data_dir}/transaction_nodes.jsonl'
edge_file = f'{data_dir}/transaction_edges.jsonl'
```

```
batch_size = 10_000

from torch_geometric.datasets import EllipticBitcoinDataset

dataset_root = f'{data_dir}/EllipticBitcoin'
dataset = EllipticBitcoinDataset(dataset_root)
data = dataset[0]

from models.helpers import upload_transaction_nodes

upload_transaction_nodes(data, node_url, node_file, batch_size)

from models.helpers import load_node_id_map, upload_transaction_edges

node_id_map = load_node_id_map(node_file)

upload_transaction_edges(data, edge_url, edge_file, batch_size, node_id_map)
```

models/gcn.py

```
import torch
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv
from torch.nn import functional as F
from torch_geometric.utils import k_hop_subgraph


class GCN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, out_channels)
        self.num_hops = sum(1 for layer in self.children() if isinstance(layer, GCNConv)) + 1
        self.optimizer = None
        self.loss_fn = None

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)
        return x

    def compile(self, optimizer, loss_fn, class_weights=None):
        self.optimizer = optimizer
        if class_weights is not None:
            class_weights_tensor = torch.tensor(class_weights,
dtype=torch.float).to(next(self.parameters()).device)
            self.loss_fn = loss_fn(weight=class_weights_tensor)
        else:
```

```
        self.loss_fn = loss_fn()

    def fit(self, data, epochs=1000):
        for epoch in range(epochs):
            self.train()
            self.optimizer.zero_grad()
            out = self(data)
            loss = self.loss_fn(out[data.train_mask], data.y[data.train_mask])
            loss.backward()
            self.optimizer.step()

            if epoch % 100 == 0:
                acc = self.evaluate(data)
                print(f'Epoch {epoch}, Loss: {loss:.4f}, Test Accuracy: {acc:.4f}')

    def predict(self, data, node_idx=None):
        self.eval()
        with torch.no_grad():
            if node_idx is not None:
                subset, edge_index, mapping, _ = k_hop_subgraph(node_idx, self.num_hops,
data.edge_index, relabel_nodes=True)
                sub_data = Data(x=data.x[subset], edge_index=edge_index)
            else:
                sub_data = data
            out = self(sub_data)
            probabilities = F.softmax(out, dim=1)
            predictions = probabilities.argmax(dim=1)
            if node_idx is not None:
                return predictions[mapping.item()], probabilities[mapping.item()]
            else:
                return predictions, probabilities

    def evaluate(self, data):
        self.eval()
        with torch.no_grad():
            out = self(data)
            pred = out.argmax(dim=1)
            correct = pred[data.test_mask] == data.y[data.test_mask]
            acc = int(correct.sum()) / int(data.test_mask.sum())
            return acc

    def load_for_inference(self, path):
        self.load_state_dict(torch.load(path))
        self.eval()
```

models/helpers.py

```
import requests
import os
import pickle
import torch
import json
```

```
from torch_geometric.data import Data


def read_n_to_last_line(filename, n=1):
    """Returns the nth before last line of a file (n=1 gives last line)
    https://stackoverflow.com/a/73195814
    """
    num_newlines = 0
    with open(filename, 'rb') as f:
        try:
            f.seek(-2, os.SEEK_END)
            while num_newlines < n:
                f.seek(-2, os.SEEK_CUR)
                if f.read(1) == b'\n':
                    num_newlines += 1
        except OSError:
            f.seek(0)
        last_line = f.readline().decode()
    return last_line


def load_node_id_map(file_path):
    node_id_map = {}
    with open(file_path, 'r') as file:
        for line in file:
            entry = json.loads(line)
            node_id_map[entry['orig_id']] = entry['new_id']
    return node_id_map


def get_node_orig_id(node_id_map, orig_id):
    return node_id_map.get(orig_id)


def upload_transaction_nodes(data, node_url, uploaded_nodes_file, batch_size):
    start_index = 0

    if os.path.exists(uploaded_nodes_file):
        last_line = read_n_to_last_line(uploaded_nodes_file, 1)
        start_index = json.loads(last_line)['orig_id'] + 1

    total_nodes = len(data.x)

    with requests.Session() as session:
        headers = {'Content-Type': 'application/json'}
        for start in range(start_index, total_nodes, batch_size):
            end = min(start + batch_size, total_nodes)
            nodes = []

            for i in range(start, end):
                node = {
                    'labels': ['Transaction'],
```

```
                'x': data.x[i].tolist(),
                'y': data.y[i].item(),
                'orig_id': i,
            }
            nodes.append(node)

        try:
            response = session.post(node_url, json=nodes, headers=headers)
            response.raise_for_status()
            new_node_ids = response.json()
            with open(uploaded_nodes_file, 'a') as file:
                for i, node in enumerate(new_node_ids):
                    json_line = json.dumps({'orig_id': start + i, 'new_id': node['id']})
                    file.write(json_line + '\n')
            print(f"Nodes {start} to {end - 1} created successfully")
        except requests.exceptions.RequestException as e:
            print(f"Failed to create nodes {start} to {end - 1}: {e}")


def upload_transaction_edges(data, edge_url, uploaded_edges_file, batch_size, node_id_map):

    edge_index = data.edge_index

    total_edges = edge_index.size(1)
    start_index = 0
    if os.path.exists(uploaded_edges_file):
        last_line = read_n_to_last_line(uploaded_edges_file, 1)
        start_index = json.loads(last_line)['orig_id'] + 1

    with requests.Session() as session:
        headers = {'Content-Type': 'application/json'}
        for start in range(start_index, total_edges, batch_size):
            end = min(start + batch_size, total_edges)
            edges = []

            for i in range(start, end):
                src_index = edge_index[0, i].item()
                dst_index = edge_index[1, i].item()

                src_id = get_node_orig_id(node_id_map, src_index)
                dst_id = get_node_orig_id(node_id_map, dst_index)

                if src_id is not None and dst_id is not None:
                    edge = {
                        'src': src_id,
                        'dst': dst_id,
                        'orig_id': i,
                    }
                    edges.append(edge)

            try:
                response = session.post(edge_url, json=edges, headers=headers)
```

```
            response.raise_for_status()
            edge_ids = response.json()
            with open(uploaded_edges_file, 'a') as file:
                for i, edge in enumerate(edge_ids):
                    json_line = json.dumps({'orig_id': start + i, 'new_id': edge['id']})
                    file.write(json_line + '\n')
            print(f"Edges {start} to {end - 1} created successfully")
        except requests.exceptions.RequestException as e:
            print(f"Failed to create edges {start} to {end - 1}: {e}")


def fetch_data(url, batch_size, page, headers={'Content-Type': 'application/json'}):
    params = {"page_size": batch_size, "page": page}
    response = requests.get(url, headers=headers, params=params)
    response.raise_for_status()
    return response.json()


def process_nodes(existing_nodes, new_nodes):
    node_id_map = {node['id']: idx for idx, node in enumerate(existing_nodes)}
    start_idx = len(existing_nodes)

    for node in new_nodes:
        if node['id'] not in node_id_map:
            node_id_map[node['id']] = start_idx
            existing_nodes.append(node)
            start_idx += 1


def process_edges(existing_edges, new_edges, existing_nodes):
    node_id_map = {node['id']: idx for idx, node in enumerate(existing_nodes)}

    for edge in new_edges:
        if edge['src'] in node_id_map and edge['dst'] in node_id_map:
            existing_edges.append(edge)


def save_data(data, file_path):
    with open(file_path, 'wb') as f:
        pickle.dump(data, f)


def load_data(file_path):
    if os.path.exists(file_path):
        with open(file_path, 'rb') as f:
            return pickle.load(f)
    return None


def fetch_and_process_graph_data(node_url, edge_url, file_path, batch_size):
    existing_nodes, existing_edges, node_page, edge_page = load_data(file_path)
```

```
    if existing_nodes is None:
        existing_nodes = []
    if existing_edges is None:
        existing_edges = []

    try:
        while True:
            node_data = fetch_data(node_url, batch_size, node_page)
            new_nodes = node_data['results']

            if not new_nodes:
                break

            process_nodes(existing_nodes, new_nodes)
            start_idx = (node_page - 1) * batch_size
            end_idx = start_idx + len(new_nodes) - 1
            print(f"Nodes {start_idx}-{end_idx} retrieved.")
            node_page += 1
            save_data((existing_nodes, existing_edges, node_page, edge_page), file_path)

        while True:
            edge_data = fetch_data(edge_url, batch_size, edge_page)
            new_edges = edge_data['results']

            if not new_edges:
                break

            process_edges(existing_edges, new_edges, existing_nodes)
            start_idx = (edge_page - 1) * batch_size
            end_idx = start_idx + len(new_edges) - 1
            print(f"Edges {start_idx}-{end_idx} retrieved.")
            edge_page += 1
            save_data((existing_nodes, existing_edges, node_page, edge_page), file_path)

    except Exception as e:
        print(f"An error occurred: {e}")
        save_data((existing_nodes, existing_edges, node_page, edge_page), file_path)
        raise

    return existing_nodes, existing_edges


def save_model(model, model_path):
    torch.save(model, model_path)


def load_model(model_path):
    return torch.load(model_path)


def load_preprocessed_data(file_path):
    with open(file_path, 'rb') as f:
```

```
        preprocessed_data = pickle.load(f)
    return preprocessed_data


def create_data_object(nodes, edges, train_mask=None, test_mask=None):
    sorted_nodes = sorted(nodes, key=lambda node: node['orig_id'])
    sorted_edges = sorted(edges, key=lambda edge: edge['orig_id'])

    node_features = []
    node_labels = []
    node_id_map = {}

    for idx, node in enumerate(sorted_nodes):
        node_id_map[node['id']] = idx
        node_features.append(node['x'])
        node_labels.append(node['y'])

    node_features = torch.tensor(node_features, dtype=torch.float)
    node_labels = torch.tensor(node_labels, dtype=torch.long)

    edge_index = []
    for edge in sorted_edges:
        if edge['src'] in node_id_map and edge['dst'] in node_id_map:
            src = node_id_map[edge['src']]
            dst = node_id_map[edge['dst']]
            edge_index.append([src, dst])

    edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    data = Data(x=node_features, edge_index=edge_index, y=node_labels, train_mask=train_mask,
test_mask=test_mask).to(device)
    return data
```

models/neo4j_to_ml.ipynb

```
import os

if os.path.basename(os.getcwd()) != 'graph':
    os.chdir('..')

from config import Config

data_dir = 'data/transaction'
base_url = Config.BASE_URL

node_url = f"{base_url}/api/nodes"
edge_url = f"{base_url}/api/edges"
batch_size = 10_000
transactions_raw_file = f'{data_dir}/transactions_raw.pkl'
```

```
transactions_file = f'{data_dir}/transactions.pkl'

from torch_geometric.datasets import EllipticBitcoinDataset

dataset_lib = EllipticBitcoinDataset(root=f'{data_dir}/EllipticBitcoin', transform=None)
data_lib = dataset_lib[0]

from models.helpers import fetch_and_process_graph_data

nodes, edges = fetch_and_process_graph_data(node_url, edge_url, transactions_raw_file,
batch_size)

from models.helpers import create_data_object, save_data

data = create_data_object(nodes, edges, data_lib.train_mask.clone(), data_lib.test_mask.clone())
save_data(data, transactions_file)

from models.transaction import TransactionClassifier

model = TransactionClassifier(in_channels=data.x.shape[1], hidden_channels=100,
out_channels=2)

model.fit(data, epochs=1000)

from models.helpers import save_model
save_model(model.state_dict(), 'models/transaction_classifier.pth')

model.predict(data)
```

models/transaction.py

```
from .gcn import GCN
import torch

class TransactionClassifier(GCN):
    def __init__(self, in_channels=165, hidden_channels=100, out_channels=2):
        super(TransactionClassifier, self).__init__(in_channels, hidden_channels, out_channels)
        class_weights = [0.3, 0.7]
        self.optimizer = torch.optim.Adam(self.parameters(), lr=1e-3, weight_decay=5e-4)
        self.compile(self.optimizer, torch.nn.CrossEntropyLoss, class_weights)
        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self.to(device)
```

openai_tools/__init__.py

```
import inspect
from typing import get_type_hints
from .transaction import predict_class, get_transaction_details

functions = {
    'transaction_get_transaction_details': get_transaction_details,
```

```
    'transaction_predict_class': predict_class
}

def generate_tool_object(func):
    signature = inspect.signature(func)
    type_hints = get_type_hints(func)
    param_descriptions = parse_docstring(func)
    parameters = []

    for name in signature.parameters.keys():
        param_type = type_hints.get(name, str)
        parameters.append({
            "name": name,
            "type": get_type_string(param_type),
            "description": param_descriptions.get(name, f"The {name} parameter.")
        })

    module_name = func.__module__.split('.')[-1]
    function_name = func.__name__
    full_name = f"{module_name}_{function_name}"

    tool_object = {
        "type": "function",
        "function": {
            "name": full_name,
            "description": func.__doc__.strip().split("\n")[0] if func.__doc__ else "No description
provided.",
            "parameters": {
                "type": "object",
                "properties": {param["name"]: {"type": param["type"], "description":
param["description"]} for param in parameters},
                "required": [param["name"] for param in parameters]
            },
        }
    }
    return tool_object


def get_type_string(annotation):
    if annotation == str:
        return "string"
    elif annotation == int:
        return "integer"
    elif annotation == list:
        return "array"
    else:
        return "string"


def parse_docstring(func):
    docstring = inspect.getdoc(func)
    param_descriptions = {}
```

```
    if docstring:
        lines = docstring.splitlines()
        for i, line in enumerate(lines):
            if line.strip().startswith("Parameters:"):
                for param_line in lines[i+1:]:
                    if not param_line.strip():
                        continue
                    if '(' in param_line and ')' in param_line:
                        param_name = param_line.split('(')[0].strip()
                        param_description = param_line.split('):')[1].strip() if '):' in param_line else ""
                        param_descriptions[param_name] = param_description
    return param_descriptions

tools = [generate_tool_object(func) for func in functions.values()]
tool_names = [tool['function']['name'] for tool in tools if tool['type'] == 'function']
```

openai_tools/transaction.py

```
from typing import List, Dict, Any
from flask import current_app


def get_transaction_details(transaction_id: str) -> Dict[str, Any]:
    """
    Get details of an transaction.

    Parameters:
    transaction_id (str): The ID of the transaction.
    """
    from db.queries import get_node
    from config import Config
    transaction = get_node(transaction_id)
    transaction['x'] = "<feature vector>"
    for field in Config.HIDDEN_FIELDS:
        if field in transaction:
            del transaction[field]
    return transaction


def predict_class(transaction_id: str) -> Dict[str, Any]:
    """
    Predict the class of a transaction. 0 for licit, 1 for illicit.

    Parameters:
    transaction_id (str): The ID of the transaction.

    Returns:
    Dict[str, Any]: A dictionary containing the predicted class.
    """
    from db.queries import get_node
    transaction = get_node(transaction_id)
```

```
    transaction_classifier = current_app.transaction_classifier
    transaction_data = current_app.transaction_data
    predictions, probabilities = transaction_classifier.predict(transaction_data, transaction['orig_id'])
    return {
        'class': predictions.item(),
        'probability': probabilities.tolist()
    }
```

routes/auth.py

```
from flask import Blueprint, request, jsonify, session
from db.queries import create_node, list_nodes
from utils.helpers import hash_password, verify_password

auth_bp = Blueprint('auth', __name__)

@auth_bp.route('/api/login', methods=['POST'])
def api_login():
    username = request.json.get('username')
    password = request.json.get('password')
    if not username or not password:
        return jsonify({'message': 'Username and password are required'}), 400

    user = list_nodes({'label': 'User', 'username': username})
    if not user:
        return jsonify({'message': 'User not found'}), 404

    hashed_password = user[0]['password']
    if not verify_password(hashed_password, password):
        return jsonify({'message': 'Incorrect password'}), 401

    session['logged_in'] = True
    session['id'] = user[0]['id']
    session['username'] = user[0]['username']
    return jsonify({'message': 'Login successful'}), 200


@auth_bp.route('/api/signup', methods=['POST'])
def api_signup():
    username = request.json.get('username')
    password = request.json.get('password')
    if not username or not password:
        return jsonify({'message': 'Username and password are required'}), 400

    if list_nodes({'label': 'User', 'username': username}):
        return jsonify({'message': 'Username already exists'}), 409

    hashed_password = hash_password(password)
    user = create_node({'username': username, 'password': hashed_password, 'labels': ['User'], 'title':
username, 'slug': 'usr'})
    session['logged_in'] = True
    session['id'] = user['id']
```

```
    session['username'] = username
    return jsonify({'message': 'Signup successful'}), 201


@auth_bp.route('/api/logout', methods=['POST'])
def api_logout():
    if not 'logged_in' in session or not session['logged_in']:
        return jsonify({'message': 'You are not logged in'}), 401
    session.clear()
    return jsonify({'message': 'Logout successful'}), 200
```

routes/files.py

```
from flask import Blueprint, request, jsonify, session, send_from_directory
from werkzeug.utils import secure_filename
import os
import uuid

files_bp = Blueprint('files', __name__)

FILE_DIR = 'static/files'
if not os.path.exists(FILE_DIR):
    os.makedirs(FILE_DIR)

@files_bp.route('/api/files/<file_id>', methods=['GET'])
def api_get_file(file_id):
    file_path = os.path.join(FILE_DIR, file_id)
    if not os.path.exists(file_path):
        return jsonify({'message': 'File not found'}), 404
    return send_from_directory(FILE_DIR, file_id, as_attachment=True)

@files_bp.route('/api/files', methods=['POST'])
def api_create_file():
    if not 'logged_in' in session or not session['logged_in']:
        return jsonify({'message': 'You do not have permission to upload files'}), 403

    file = request.files.get('file')
    if not file:
        return jsonify({'message': 'File is required'}), 400

    original_filename = secure_filename(file.filename)
    filename_base, file_extension = os.path.splitext(original_filename)

    random_uuid = uuid.uuid4().hex
    new_filename = f"{filename_base}_{random_uuid}{file_extension}"

    file_path = os.path.join(FILE_DIR, new_filename)
    file.save(file_path)

    return jsonify({'id': new_filename}), 201

@files_bp.route('/api/files/<file_id>', methods=['DELETE'])
```

```
def api_delete_file(file_id):
    if not 'logged_in' in session or not session['logged_in']:
        return jsonify({'message': 'You do not have permission to delete files'}), 403

    file_path = os.path.join(FILE_DIR, file_id)
    if not os.path.exists(file_path):
        return jsonify({'message': 'File not found'}), 404

    os.remove(file_path)
    return jsonify({'message': 'File deleted successfully'}), 200
```

routes/graph.py

```
from flask import Blueprint, request, jsonify
from db.queries import get_node, list_nodes, create_node, create_node_batch, list_node_labels,
update_node, delete_node, create_index, get_edge, create_edge, create_edge_batch, list_edges
from config import Config
import json

graph_bp = Blueprint('graph', __name__)

@graph_bp.route('/api/nodes/<id>', methods=['GET'])
def api_get_node(id):
    node = get_node(id)
    if node is None:
        return jsonify({'message': 'Node not found'}), 404
    for field in Config.HIDDEN_FIELDS:
        if field in node:
            del node[field]
    return node


@graph_bp.route('/api/nodes', methods=['GET'])
def api_list_nodes():
    page = int(request.args.get('page', 1))
    page_size = int(request.args.get('page_size', 10))
    start_date = request.args.get('start_date')
    end_date = request.args.get('end_date')
    sort_by = request.args.get('sort_by')
    sort_order = request.args.get('sort_order')
    query = request.args.get('q', "")
    filters = {}
    if 'filter' in request.args:
        filters = json.loads(request.args['filter'])

    nodes = list_nodes(filters=filters, query=query, page=page, page_size=page_size,
start_date=start_date, end_date=end_date, sort_by=sort_by, sort_order=sort_order)
    for node in nodes:
        for field in Config.HIDDEN_FIELDS:
            if field in node:
                del node[field]
```

```
    return jsonify({"results": nodes})


@graph_bp.route('/api/nodes', methods=['POST'])
def api_create_node():
    properties = request.json

    if 'labels' not in properties:
        return jsonify({'message': 'Label array is required'}), 400

    reserved_fields_used = set(properties.keys()).intersection(Config.RESERVED_FIELDS)
    if reserved_fields_used:
        return jsonify({'message': f'Reserved fields used: {reserved_fields_used}'}), 400

    reserved_labels_used = set(properties['labels']).intersection(Config.RESERVED_LABELS)
    if reserved_labels_used:
        return jsonify({'message': f'Reserved labels used: {reserved_labels_used}'}), 400

    node = create_node(properties)
    if node is None:
        return jsonify({'message': 'Node not created'}), 500
    return jsonify(node), 201


@graph_bp.route('/api/nodes/<node_id>', methods=['PUT'])
def api_update_node(node_id):
    properties = request.json
    labels = properties.get('labels', [])

    reserved_fields_used = set(properties.keys()).intersection(Config.RESERVED_FIELDS)
    if reserved_fields_used:
        return jsonify({'message': f'Reserved fields used: {reserved_fields_used}'}), 400

    reserved_labels_used = set(labels).intersection(Config.RESERVED_LABELS)
    if reserved_labels_used:
        return jsonify({'message': f'Reserved labels used: {reserved_labels_used}'}), 400

    node = update_node(node_id, properties)
    if node is None:
        return jsonify({'message': 'Node not found'}), 404
    return jsonify(node), 200


@graph_bp.route('/api/nodes/<node_id>', methods=['DELETE'])
def api_delete_node(node_id):
    node = delete_node(node_id)
    if node is None:
        return jsonify({'message': 'Node not found'}), 404
    return jsonify({'message': 'Node deleted successfully'}), 200


@graph_bp.route('/api/nodes/batch', methods=['POST'])
```

```
def api_create_node_batch():
    nodes = request.json

    if not isinstance(nodes, list):
        return jsonify({'message': 'A list of node data is required'}), 400

    for properties in nodes:
        if 'labels' not in properties:
            return jsonify({'message': 'Label array is required'}), 400

        reserved_fields_used = set(properties.keys()).intersection(Config.RESERVED_FIELDS)
        if reserved_fields_used:
            return jsonify({'message': f'Reserved fields used: {reserved_fields_used}'}), 400

        reserved_labels_used = set(properties['labels']).intersection(Config.RESERVED_LABELS)
        if reserved_labels_used:
            return jsonify({'message': f'Reserved labels used: {reserved_labels_used}'}), 400

    result = create_node_batch(nodes)
    if result is None:
        return jsonify({'message': 'Nodes not created'}), 500
    return jsonify(result), 201


@graph_bp.route('/api/edges/<id>', methods=['GET'])
def api_get_edge(id):
    edge = get_edge(id)
    if edge is None:
        return jsonify({'message': 'Edge not found'}), 404
    for field in Config.HIDDEN_FIELDS:
        if field in edge:
            del edge[field]
    return edge


@graph_bp.route('/api/edges', methods=['GET'])
def api_list_edges():
    page = int(request.args.get('page', 1))
    page_size = int(request.args.get('page_size', 10))
    start_date = request.args.get('start_date')
    end_date = request.args.get('end_date')
    sort_by = request.args.get('sort_by')
    sort_order = request.args.get('sort_order')
    filters = {}
    if 'filter' in request.args:
        filters = json.loads(request.args['filter'])

    edges = list_edges(filters=filters, page=page, page_size=page_size, start_date=start_date,
end_date=end_date, sort_by=sort_by, sort_order=sort_order)
    for edge in edges:
        for field in Config.HIDDEN_FIELDS:
            if field in edge:
```

```
            del edge[field]
    return jsonify({"results": edges})


@graph_bp.route('/api/edges', methods=['POST'])
def api_create_edge():
    data = request.json
    if 'src' not in data or 'dst' not in data:
        return jsonify({'message': 'Source and destination are required'}), 400
    reserved_fields_used = set(data.keys()).intersection(Config.RESERVED_FIELDS)
    if reserved_fields_used:
        return jsonify({'message': f'Reserved fields used: {reserved_fields_used}'}), 400
    edge = create_edge(data)
    if edge is None:
        return jsonify({'message': 'Edge not created'}), 500
    return jsonify(edge), 201


@graph_bp.route('/api/edges/batch', methods=['POST'])
def api_create_edge_batch():
    edges = request.json
    if not isinstance(edges, list):
        return jsonify({'message': 'A list of edge data is required'}), 400
    for data in edges:
        if 'src' not in data or 'dst' not in data:
            return jsonify({'message': 'Source and destination are required'}), 400
        reserved_fields_used = set(data.keys()).intersection(Config.RESERVED_FIELDS)
        if reserved_fields_used:
            return jsonify({'message': f'Reserved fields used: {reserved_fields_used}'}), 400
    result = create_edge_batch(edges)
    if result is None:
        return jsonify({'message': 'Edges not created'}), 500
    return jsonify(result), 201


@graph_bp.route('/api/labels', methods=['GET'])
def api_list_node_labels():
    labels = list_node_labels()
    return jsonify({"labels": labels})
```

routes/main.py

```
from flask import Blueprint, render_template, redirect, url_for, request, session, jsonify
from db.queries import list_nodes, get_node, list_node_labels, get_graph_neighborhood
from utils.helpers import create_openai_completion, create_openai_transcription
import json
from config import Config

main_bp = Blueprint('main', __name__)
```

```
@main_bp.route('/', methods=['GET'])
def index():
    recent_items = list_nodes()
    labels = list_node_labels()
    data = {
        'graph': get_graph_neighborhood([item['id'] for item in recent_items]),
        'labels': labels,
    }
    return render_template('index.html', data=data)


@main_bp.route('/search', methods=['GET'])
def search():
    page = int(request.args.get('page', 1))
    page_size = int(request.args.get('page_size', 10))
    query = request.args.get('q', '')
    start_date = request.args.get('start_date')
    end_date = request.args.get('end_date')
    insights = request.args.get('insights') == 'true'
    filters = request.args.get('filter', '{}')
    filters = json.loads(filters)

    labels = list_node_labels()
    nodes = list_nodes(filters=filters, query=query, page=page, page_size=page_size,
start_date=start_date, end_date=end_date)
    for node in nodes:
        for field in Config.HIDDEN_FIELDS:
            if field in node:
                del node[field]

    data = {
        'results': nodes,
        'graph': get_graph_neighborhood([node['id'] for node in nodes]),
        'labels': labels,
    }
    return render_template('search.html', data=data, insights=insights)


@main_bp.route('/items/<item_id>', methods=['GET'])
def get_item(item_id):
    node = get_node(item_id)
    for field in Config.HIDDEN_FIELDS:
            if field in node:
                del node[field]
    if node is None:
        return redirect(url_for('main.index'))
    data = {
        'item': node,
        'labels': list_node_labels(),
        'graph': get_graph_neighborhood([item_id])
    }
    return render_template('get_item.html', data=data)
```

```
@main_bp.route('/new', methods=['GET'])
def create_item():
    if not 'logged_in' in session or not session['logged_in']:
        return redirect(url_for('main.index'))
    labels = list_node_labels()
    return render_template('create_item.html', data={"labels": labels})



@main_bp.route('/api/chat/completions', methods=['POST'])
def api_create_completion():
    messages = request.json.get('messages')
    return create_openai_completion(messages)


@main_bp.route('/api/audio/transcriptions', methods=['POST'])
def api_create_transcription():
    file = request.files.get('file')
    if not file:
        return jsonify({'message': 'File is required'}), 400
    allowed_formats = ['flac', 'm4a', 'mp3', 'mp4', 'mpeg', 'mpga', 'oga', 'ogg', 'wav', 'webm']
    file_extension = file.filename.split('.')[-1].lower()
    if file_extension not in allowed_formats:
        return jsonify({'message': f"Unsupported file format. Supported formats: {allowed_formats},
got '{file_extension}'"}), 400
    text = create_openai_transcription(file)
    return jsonify({'text': text})
```

routes/openai_tools.py

```
from flask import Blueprint, request, jsonify
from utils.helpers import list_tools, use_tool

openai_tools_bp = Blueprint('openai_tools', __name__)


@openai_tools_bp.route('/api/tools', methods=['GET'])
def api_list_tools():
    tools = list_tools()
    return jsonify({'tools': tools})


@openai_tools_bp.route('/api/tools', methods=['POST'])
def api_use_tool():
    data = request.json
    tool_name = data.get('name')
    tool_arguments = data.get('arguments')
    if not tool_name:
        return jsonify({'message': 'Tool name is required'}), 400
    result = use_tool(tool_name, tool_arguments)
```

```
    if result.get('error'):
        return jsonify(result), 400
    return jsonify(result), 200
```

scripts/base.js

```
function validateSearchForm() {
    const query = document.querySelector('#input-query').value;
    const isAIAssistEnabled = document.querySelector('#input-insights').checked;
    const alertSearch = document.querySelector('#alert-search');

    if (!query) {
        alertSearch.classList.remove('d-none');
        return false;
    }

    alertSearch.classList.add('d-none');
    const searchURL = `/search?q=${encodeURIComponent(query)}${isAIAssistEnabled ?
'&insights=true' : ''}`;

    window.location.href = searchURL;
    return false;
}


function serializeForm(formId) {
    const form = document.getElementById(formId);
    const formData = new FormData(form);
    const json = {};
    formData.forEach((value, key) => {
        json[key] = value || "";
    });
    return json;
}

function validateSignupForm() {
    const json = serializeForm('form-signup');
    fetch(url_signup, {
        method: 'POST',
        body: JSON.stringify(json),
        headers: {
            'Content-Type': 'application/json'
        }
    })
        .then(response => {
            if (response.ok) {
                document.getElementById('message-signup-error').style.display = 'none';
                return response.json();
            } else {
                document.getElementById('message-signup-success').style.display = 'none';
                return response.json().then(data => {
```

```
                throw new Error(data.message);
            });
        }
    })
    .then(() => {
        location.reload();
    })
    .catch(error => {
        const errorMessage = document.getElementById('message-signup-error');
        errorMessage.textContent = error.message;
        errorMessage.style.display = 'block';
    });

  return false;
}

function validateLoginForm() {
  const json = serializeForm('form-login');
  fetch(url_login, {
    method: 'POST',
    body: JSON.stringify(json),
    headers: {
      'Content-Type': 'application/json'
    }
  })
    .then(response => {
      if (response.ok) {
        document.getElementById('modal-login').style.display = 'none';
        document.getElementById('message-login-error').style.display = 'none';
        location.reload();
      } else {
        return response.json().then(data => {
          throw new Error(data.message);
        });
      }
    })
    .catch(error => {
      const errorMessage = document.getElementById('message-login-error');
      errorMessage.textContent = error;
      errorMessage.style.display = 'block';
    });

  return false;
}

function logOut() {
  fetch(url_logout, {
    method: 'POST'
  })
  .then((response) => {
    if (response.ok) {
      location.reload();
```

```
        } else {
            return response.json().then(data => {
                throw new Error(data.message);
            });
        }
    })
    .catch(error => {
        console.error(error);
    });
}


const loginLink = document.getElementById('link-login-modal');
const signupLink = document.getElementById('link-signup-modal');
const loginModal = new bootstrap.Modal('#modal-login', {
    keyboard: false
})
const signupModal = new bootstrap.Modal('#modal-signup', {
    keyboard: false
})

signupLink.addEventListener('click', function () {
    loginModal.hide();
    signupModal.show();
});

loginLink.addEventListener('click', function () {
    signupModal.hide();
    loginModal.show();
});
```

scripts/create_item.js

```
setElementVisibility({
    '#btn-go-create-item': false,
    '#btn-back': true,
    '#btn-delete-item': false,
    '#btn-cancel-item': false,
    '#btn-edit-item': false,
    '#btn-save-item': false,
    '#btn-create-item': true
});

function generateRandomPlaceholder() {
    const placeholders = [
        {
            "title": "May the Forms Be With You",
            "content": "A guide on how to efficiently fill out and manage digital forms in your business
processes."
        },
        {
```

```
            "title": "Winter is Coming",
            "content": "An advisory piece on how businesses can prepare for the economic downturn or
seasonal business fluctuations."
        },
        {
            "title": "You Shall Not Pass… Without Authentication",
            "content": "A detailed guide on the importance of cybersecurity measures and authentication
protocols."
        },
        {
            "title": "To Infinity and Beyond!",
            "content": "An inspirational piece on pushing the limits of innovation and achieving the
seemingly impossible in technology."
        }
    ];
    const randomIndex = Math.floor(Math.random() * placeholders.length);
    return placeholders[randomIndex];
}

if (inputItemTitle && inputItemContent) {
    placeholder = generateRandomPlaceholder();
    inputItemTitle.placeholder = placeholder.title;
    inputItemContent.setAttribute('placeholder', placeholder.content);
}
```

scripts/get_item.js

```
setElementVisibility({
    '#btn-back': true,
    '#btn-delete-item': true,
    '#btn-cancel-item': false,
    '#btn-edit-item': true,
    '#btn-save-item': false,
    '#btn-create-item': false,
    '#label-input-item-title': false,
    '#input-item-title': false,
});

disableEditor();
```

scripts/search.js

```
function updateProgressBars() {
    const progressBars = document.querySelectorAll('.progress-bar');
    progressBars.forEach(function (bar) {
        const score = parseFloat(bar.getAttribute('data-score'));
        const percentage = score * 100;
        let progressClass;
        let progressText;

        if (score < 0.5) {
```

```
            progressClass = 'bg-danger';
            progressText = 'Bad Match'
        } else if (score < 0.75) {
            progressClass = 'bg-warning';
            progressText = 'Average Match';
        } else {
            progressClass = 'bg-success';
            progressText = 'Close Match';
        }

        bar.style.width = percentage.toFixed(2) + '%';
        bar.classList.add(progressClass);
        bar.setAttribute('aria-valuenow', score);
        bar.innerText = progressText;
    });
}

function formatDateTime() {
    document.querySelectorAll('.datetime').forEach(function (element) {
        const utcTime = element.getAttribute('data-datetime');
        const date = new Date(utcTime);

        const options = {
            year: 'numeric',
            month: 'long',
            day: 'numeric',
            hour: 'numeric',
            minute: 'numeric',
            hour12: true
        };
        const userLocale = navigator.language || 'en-US';
        const formattedDate = new Intl.DateTimeFormat(userLocale, options).format(date);
        element.innerHTML = formattedDate;
    });
}


async function getInsights() {
    const insightsContent = document.querySelector('#insights-content');
    const urlParams = new URLSearchParams(window.location.search);
    const query = urlParams.get('q', '');

    const systemMessage = {
        role: 'system',
        content: "You are a helpful assistant that provides insights into information that is stored in a
knowledge base. You can call external functions to retrieve additional data if necessary."
    }

    const userMessage = {
        role: 'user',
        content: query
    };
```

```
    const messages = [systemMessage, userMessage];

    const response = await fetch(url_create_completion, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            messages
        })
    });

    const reader = response.body.getReader();
    const decoder = new TextDecoder();
    let content = '';
    while (true) {
        const { done, value } = await reader.read();
        content += decoder.decode(value);
        insightsContent.innerHTML = marked.parse(content);
        if (done) {
            break;
        }
    }
}


function initializeSearch() {
    updateProgressBars();
    formatDateTime();

    const urlParams = new URLSearchParams(window.location.search);
    const insights = urlParams.get('insights') === 'true';
    if (insights) {
        getInsights();
    }

}


initializeSearch()
```

static/trix.js

```
const btnBack = document.querySelector('#btn-back');
const btnDelete = document.querySelector('#btn-delete-item');
const btnCancel = document.querySelector('#btn-cancel-item');
const btnEdit = document.querySelector('#btn-edit-item');
const btnSave = document.querySelector('#btn-save-item');
const btnCreate = document.querySelector('#btn-create-item');

const formItem = document.querySelector('#form-item');
```

```
const itemTitle = document.querySelector('#item-title');
const inputItemTitle = document.querySelector('#input-item-title');
const inputItemContent = document.querySelector('#input-item-content');
const inputHiddenTrix = document.querySelector('#input-hidden-trix');
const alertSubmissionSuccess = document.querySelector('#alert-submission-success');
const spinnerLoading = document.querySelector('#spinner-loading');
const itemActions = document.querySelector('#item-actions');

const trixToolbar = document.querySelector('trix-toolbar');

const redirectTimeout = 1500;

function enableEditor() {
    inputItemTitle.removeAttribute('readonly');
    inputItemContent.editor.element.setAttribute('contentEditable', true)
    trixToolbar.classList.remove('d-none');
}

function disableEditor() {
    inputItemTitle.setAttribute('readonly', '');
    inputItemContent.editor.element.setAttribute('contentEditable', false)
    trixToolbar.classList.add('d-none');
}

function validateItemForm() {
    const title = document.querySelector('#input-item-title').value;
    const content = document.querySelector('#input-item-content').value;

    const alertContentMissing = document.querySelector('#alert-content-missing');

    if (!title || !content) {
        alertContentMissing.classList.remove('d-none');
        return false;
    }

    alertContentMissing.classList.add('d-none');

    return true;
}

function createItem(title, content) {
    if (!validateItemForm()) {
        return false;
    }

    inputItemTitle.setAttribute('readonly', '');
    inputItemContent.editor.element.setAttribute('contentEditable', false)
    btnCreate.classList.add('d-none');
    btnBack.classList.add('d-none');
    spinnerLoading.classList.remove('d-none');

    const labels = [];
```

```
    const labelInputs = document.querySelectorAll('input[name="labels"]:checked');

    labelInputs.forEach((labelInput) => {
        labels.push(labelInput.value);
    });

    fetch(`${url_create_item}`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            title,
            content,
            labels
        })
    })
        .then(response => {
            if (response.ok) {
                alertSubmissionSuccess.classList.remove('d-none');
                setInterval(() => {
                    history.back();
                }, redirectTimeout);
            } else {
                return response.json().then(data => {
                    throw new Error(data.message);
                });
            }
        })
        .catch(error => {
            const alertSubmissionError = document.querySelector('#alert-submission-error');
            alertSubmissionError.textContent = error.message;
            alertSubmissionError.classList.remove('d-none');

            inputItemTitle.removeAttribute('readonly');
            inputItemContent.editor.element.setAttribute('contentEditable', true)
            btnCreate.classList.remove('d-none');
            btnBack.classList.remove('d-none');
            spinnerLoading.classList.add('d-none');
        })
        .finally(() => {
            spinnerLoading.classList.add('d-none');
        });
}

function updateItem(itemId, title, content) {
    if (!validateItemForm()) {
        return false;
    }
    url = url_update_item.replace('NODE_ID', itemId);

    disableEditor();
```

```
    setElementVisibility({
        '#btn-back': false,
        '#btn-delete-item': false,
        '#btn-cancel-item': false,
        '#btn-edit-item': false,
        '#btn-save-item': false,
        '#btn-create-item': false
    });

    spinnerLoading.classList.remove('d-none');

    fetch(`${url}`, {
        method: 'PUT',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            title: title,
            content: content
        })
    })
        .then(response => {
            if (response.ok) {
                alertSubmissionSuccess.classList.remove('d-none');
                setInterval(() => {
                    history.back();
                }, redirectTimeout);
            } else {
                return response.json().then(data => {
                    throw new Error(data.message);
                });
            }
        })
        .catch(error => {
            const alertSubmissionError = document.querySelector('#alert-submission-error');
            alertSubmissionError.textContent = error.message;
            alertSubmissionError.classList.remove('d-none');
            enableEditor();
            btnCancel.classList.remove('d-none');
            btnSave.classList.remove('d-none');
            trixToolbar.classList.remove('d-none');
        })
        .finally(() => {
            spinnerLoading.classList.add('d-none');
        });
}

function deleteItem(itemId) {
    url = url_delete_item.replace('NODE_ID', itemId);

    btnBack.classList.add('d-none');
```

```
    btnEdit.classList.add('d-none');
    btnDelete.classList.add('d-none');
    btnSave.classList.add('d-none');
    btnCancel.classList.add('d-none');
    btnCreate.classList.add('d-none');

    spinnerLoading.classList.remove('d-none');

    fetch(`${url}`, {
        method: 'DELETE'
    })
        .then(response => {
            if (response.ok) {
                alertSubmissionSuccess.classList.remove('d-none');
                setInterval(() => {
                    location.href = '/'
                }, redirectTimeout);
            } else {
                return response.json().then(data => {
                    throw new Error(data.message);
                });
            }
        })
        .catch(error => {
            const alertSubmissionError = document.querySelector('#alert-submission-error');
            alertSubmissionError.textContent = error.message;
            setElementVisibility({
                '#alert-submission-error': true,
                '#btn-back': true,
                '#btn-delete-item': true,
                '#btn-edit-item': true
            });
        })
        .finally(() => {
            spinnerLoading.classList.add('d-none');
        });
}

if (btnBack) {
    btnBack.addEventListener('click', function() {
        history.back();
    });
}

if (btnDelete) {
    btnDelete.addEventListener('click', function () {
        itemId = formItem.getAttribute('data-item-id');
        deleteItem(itemId);
    });
}

if (btnCancel) {
```

```
    btnCancel.addEventListener('click', function () {
        setElementVisibility({
            '#btn-back': true,
            '#btn-delete-item': true,
            '#btn-cancel-item': false,
            '#btn-edit-item': true,
            '#btn-save-item': false,
            '#btn-create-item': false,
            '#label-input-item-title': false,
            '#input-item-title': false,
            '#item-title': true,
            'trix-toolbar': false
        });

        inputItemTitle.setAttribute('readonly', true);
        inputItemContent.editor.element.setAttribute('contentEditable', false);
    });
}

if (btnEdit) {
    btnEdit.addEventListener('click', function() {
        enableEditor();
        inputItemContent.focus();
        setElementVisibility({
            '#btn-back': false,
            '#btn-delete-item': false,
            '#btn-cancel-item': true,
            '#btn-edit-item': false,
            '#btn-save-item': true,
            '#btn-create-item': false,
            '#label-input-item-title': true,
            '#input-item-title': true,
            '#item-title': false
        });
    });
}

if (btnSave) {
    btnSave.addEventListener('click', function() {
        itemId = formItem.getAttribute('data-item-id');
        title = inputItemTitle.value;
        content = inputItemContent.value;
        updateItem(itemId, title, content);
    });
}

if (btnCreate) {
    btnCreate.addEventListener('click', function() {
        title = inputItemTitle.value;
        content = inputItemContent.value;
        createItem(title, content);
    });
```

```
}


document.addEventListener('trix-attachment-add', function (event) {
    console.log('trix-attachment-add');
    console.log(event);
    const attachment = event.attachment;
    uploadAttachment(attachment);
});


document.addEventListener('trix-attachment-remove', function (event) {
    console.log('trix-attachment-remove');
    console.log(event);
    const attachment = event.attachment;
    if (attachment.file) {
        return deleteAttachment(attachment);
    }
});


function prepareFormData(file) {
    const form = new FormData();
    form.append('file', file);
    return form;
}

async function uploadFile(form) {
    const response = await fetch(url_create_file, {
        method: 'POST',
        body: form
    });
    if (!response.ok) {
        throw new Error('Error uploading file');
    }
    return await response.json();
}

function handleFileUploadResponse(data, file, selectedRange, attachment) {
    const url = url_get_file.replace('FILE_ID', data.id);
    attachment.setAttributes({
        url: url
    });

    if (file.type.startsWith('audio')) {
        console.log('Processing audio file');
        return processAudioFile(url, selectedRange);
    } else if (file.type.startsWith('image')) {
        console.log('Processing image file');
        return processImageFile(url, attachment);
    }
}
```

```
function processAudioFile(url, selectedRange) {
    const audioHtml = `<audio controls src=${url}>`;
    const audioAttachment = new Trix.Attachment({ content: audioHtml });
    inputItemContent.editor.setSelectedRange(selectedRange);
    inputItemContent.editor.insertAttachment(audioAttachment);
}

function createAudioTranscription(form, file) {
    fetch(url_create_transcription, {
        method: 'POST',
        body: form
    }).then(response => {
        if (!response.ok) {
            throw new Error('Error creating transcription');
        }
        return response.json();
    }).then(data => {
        inputItemContent.editor.insertHTML(`<i>Audio transcription:\n${data.text}</i>`);
    }).catch(console.error);
}

function processImageFile(url, attachment) {
    return fetch(url_create_completion, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            messages: [{
                role: 'user',
                content: [
                    { type: 'text', text: "What's in this image?" },
                    { type: 'image_url', image_url: { url: `${window.location.origin}${url}` } }
                ]
            }]
        })
    }).then(response => processImageResponse(response, attachment))
        .catch(console.error);
}

function processImageResponse(response, attachment) {
    const reader = response.body.getReader();
    const decoder = new TextDecoder();
    let accumulatedDescription = "";
    readStreamToUpdateCaption(reader, decoder, attachment, accumulatedDescription);
}

function readStreamToUpdateCaption(reader, decoder, attachment, accumulatedDescription) {
    reader.read().then(({ done, value }) => {
        if (done) {
            return;
```

```
        }
        accumulatedDescription += decoder.decode(value);
        readStreamToUpdateCaption(reader, decoder, attachment, accumulatedDescription);
        updateImageCaption(accumulatedDescription, attachment);
    });
}

function updateImageCaption(description, attachment) {
    const editor = document.querySelector('trix-editor').editor;
    const originalRange = editor.getSelectedRange();
    const attachmentRange = editor.getDocument().getRangeOfAttachment(attachment);

    editor.setSelectedRange(attachmentRange);
    editor.activateAttribute("caption", description);
    editor.setSelectedRange(originalRange);
}

function readStream(reader, decoder) {
    reader.read().then(({ done, value }) => {
        if (done) {
            return;
        }
        inputItemContent.editor.insertHTML(`<i>${decoder.decode(value)}</i>`);
        readStream(reader, decoder);
    });
}

function restoreEditorState() {
    inputItemContent.editor.element.setAttribute('contentEditable', true);
}

function uploadAttachment(attachment) {
    const file = attachment.file;
    const selectedRange = inputItemContent.editor.getSelectedRange();

    if (file) {
        const form = prepareFormData(file);
        inputItemContent.editor.element.setAttribute('contentEditable', false);

        const uploadFilePromise = uploadFile(form)
            .then(data => handleFileUploadResponse(data, file, selectedRange, attachment))
            .catch(console.error);

        const transcriptionPromise = file.type.startsWith('audio') ?
            createAudioTranscription(form, file) : Promise.resolve();

        Promise.all([uploadFilePromise, transcriptionPromise])
            .catch(console.error)
            .finally(restoreEditorState);
    }
}
```

```
function deleteAttachment(attachment) {
    const fileId = attachment.attachment.attributes.values.url.split('/').pop();
    const url = url_delete_file.replace('FILE_ID', fileId);
    fetch(url, {
        method: 'DELETE'
    })
        .then(response => {
            if (response.ok) {
                return response.json();
            } else {
                throw new Error('Error deleting file');
            }
        })
        .catch(error => {
            console.error(error);
        })
}
```

utils/helpers.py

```
from bs4 import BeautifulSoup
from openai import OpenAI
from config import Config
from io import BytesIO
from db.neo4j_connection import conn
import shortuuid
from werkzeug.security import generate_password_hash, check_password_hash
from flask_caching import Cache
from openai_tools import tools, tool_names, functions
import json
from flask import stream_with_context, current_app

openai_client = OpenAI(api_key=Config.OPENAI_API_KEY)

def init_cache(app):
    app.cache = Cache(app, config={
        'CACHE_TYPE': 'FileSystemCache',
        'CACHE_DIR': 'data/cache',
        'CACHE_DEFAULT_TIMEOUT': 86400
    })


def check_cache(cache_key):
    return current_app.cache.get(cache_key)


def create_openai_embedding(input):
    cache_key = f'openai_embedding_{input}'
    cached_data = check_cache(cache_key)
    if cached_data:
```

```
        return cached_data
    embedding = openai_client.embeddings.create(
        input=input,
        model=Config.OPENAI_EMBEDDING_MODEL
    ).data[0].embedding
    current_app.cache.set(cache_key, embedding)
    return embedding


def create_item_embedding(item):
    return create_openai_embedding(f"Title: {item['title']}\nLabels: {',
'.join(item['labels'])}\nContent: {item['content']}")


def create_openai_completion(messages):
    def generate():
        stream = openai_client.chat.completions.create(
            model=Config.OPENAI_COMPLETION_MODEL,
            messages=messages,
            tools=tools,
            tool_choice='auto',
            stream=True
        )
        streaming_content = ""
        tool_calls = []
        for chunk in stream:
            msg = chunk.choices[0]
            delta = msg.delta
            if delta.content is not None:
                streaming_content += delta.content
                yield delta.content
            if delta.tool_calls is not None:
                tc_chunks = delta.tool_calls
                for tc_chunk in tc_chunks:
                    if len(tool_calls) <= tc_chunk.index:
                        tool_calls.append({"id": "", "type": "function", "function": { "name": "",
"arguments": "" } })
                    tc = tool_calls[tc_chunk.index]

                    if tc_chunk.id:
                        tc["id"] += tc_chunk.id
                    if tc_chunk.function.name:
                        tc["function"]["name"] += tc_chunk.function.name
                    if tc_chunk.function.arguments:
                        tc["function"]["arguments"] += tc_chunk.function.arguments
            if msg.finish_reason == "stop":
                messages.append({"role": "assistant", "content": streaming_content})
            elif msg.finish_reason == "tool_calls":
                messages.append(
                    {
                        "tool_calls": tool_calls,
                        "role": "assistant",
```

```
                    "content": None
                }
            )
            for tool_call in tool_calls:
                before_tool_call = f"<div class='alert alert-primary' role='alert'>Triggered  <span
class='fw-bold'>{tool_call['function']['name']}</span> with arguments
{tool_call['function']['arguments']}</div>"
                yield before_tool_call
                function_output = json.dumps(use_tool(
                    tool_call["function"]["name"], json.loads(tool_call["function"]["arguments"])
                ))
                messages.append(
                    {
                        "tool_call_id": tool_call["id"],
                        "role": "tool",
                        "content": f"{function_output}"
                    }
                )
            yield from create_openai_completion(
                messages
            )

    return stream_with_context(generate())


def create_openai_transcription(file):
    buffer = BytesIO(file.read())
    buffer.name = file.filename
    transcript = openai_client.audio.transcriptions.create(
        model="whisper-1",
        file=buffer
    )
    return transcript.text


def html_to_text(html_content):
    soup = BeautifulSoup(html_content, "html.parser")
    return soup.get_text()


def hash_password(password):
    return generate_password_hash(password)


def verify_password(hashed_password, input_password):
    return check_password_hash(hashed_password, input_password)


def convert_results(results):
    nodes = []
    links = []
    result_data = []
```

```
    node_ids = set()

    for result in results:
        if result['user_id'] not in node_ids:
            nodes.append({"title": result['user_name'], "id": result['user_id'], "label": "User"})
            node_ids.add(result['user_id'])

        for doc in result['items']:
            if doc['id'] not in node_ids:
                nodes.append({"title": doc['title'], "id": doc['id'], "label": "Document"})
                node_ids.add(doc['id'])
            links.append({"source": doc['id'], "target": result['user_id']})
            result_data.append({
                "title": doc['title'],
                "id": doc['id'],
                "content": doc['content'],
                "created_at": doc['created_at'],
                "author": doc['author'],
                "score": doc.get('score', 1),
            })

    data = {
        "graph": {
            "nodes": nodes,
            "links": links
        },
        "results": result_data
    }

    return data


def get_documents_data(query):
    if not query:
        return None

    openai_response = create_openai_embedding(query)

    neo4j_query = """
        CALL db.index.vector.queryNodes('document-embeddings', $num_results, $embedding)
YIELD node AS similarDocument, score
        MATCH (u:User)-[r:CREATED]->(d:Document)
        WHERE d = similarDocument
        WITH u, d, r, score
        ORDER BY score DESC
        WITH u, COLLECT({
            id: elementId(d),
            title: d.title,
            content: d.content,
            created_at: d.created_at,
            author: u.username,
```

```
            rel_type: type(r),
            score: score
        }) AS documents
        RETURN u.username AS user_name, elementId(u) AS user_id, documents
    """

    results = conn.query(
        query=neo4j_query,
        parameters={
            'embedding': openai_response.data[0].embedding,
            'num_results': 5
        },
        db="neo4j"
    )
    data = convert_results(results)
    data['query'] = query
    return data


def generate_id(labels, slug=None, length=16):
    if slug is None and labels:
        slug = labels[0][:3].lower()
    elif slug is None:
        slug = "itm"
    return f"{slug}_{shortuuid.ShortUUID().random(length)}"


def list_tools():
    return tools


def use_tool(tool_name, tool_arguments):
    if tool_name in tool_names:
        tool = functions[tool_name]
        return tool(**tool_arguments)
    else:
        return {"error": "Tool not found"}
```

app.py

```
from flask import Flask
from waitress import serve
import os
from dotenv import load_dotenv

load_dotenv()

app = Flask(__name__)
app.secret_key = os.environ.get("SECRET_KEY")
```

```
from utils.helpers import init_cache
init_cache(app)

from models.transaction import TransactionClassifier
app.transaction_classifier = TransactionClassifier()
app.transaction_classifier.load_for_inference('models/transaction_classifier.pth')

from models.helpers import load_preprocessed_data
app.transaction_data = load_preprocessed_data('data/transaction/transactions.pkl')

from routes.main import main_bp
from routes.auth import auth_bp
from routes.files import files_bp
from routes.graph import graph_bp
from routes.openai_tools import openai_tools_bp

app.register_blueprint(main_bp)
app.register_blueprint(auth_bp)
app.register_blueprint(files_bp)
app.register_blueprint(graph_bp)
app.register_blueprint(openai_tools_bp)

if __name__ == '__main__':
    serve(app, host='0.0.0.0', port=5004)
```

config.py

```
import os

class Config:
    SECRET_KEY = os.environ.get("SECRET_KEY")
    BASE_URL = os.environ.get("BASE_URL")
    OPENAI_API_KEY = os.environ.get("OPENAI_API_KEY")
    OPENAI_EMBEDDING_MODEL = os.environ.get("OPENAI_EMBEDDING_MODEL")
    OPENAI_COMPLETION_MODEL = os.environ.get("OPENAI_COMPLETION_MODEL")
    OPENAI_TRANSCRIPTION_MODEL =
os.environ.get("OPENAI_TRANSCRIPTION_MODEL")
    NEO4J_URI = os.environ.get("NEO4J_URI")
    NEO4J_ADMIN_USERNAME = os.environ.get("NEO4J_ADMIN_USERNAME")
    NEO4J_ADMIN_PASSWORD = os.environ.get("NEO4J_ADMIN_PASSWORD")
    OPENAI_EMBEDDING_DIMENSIONS =
os.environ.get("OPENAI_EMBEDDING_DIMENSIONS")
    RESERVED_FIELDS = ['id', 'created_at', 'updated_at', 'created_by', 'updated_by', 'username',
'password', 'embedding']
    HIDDEN_FIELDS = ['password', 'embedding']
    RESERVED_LABELS = ['User']
```