ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

КАФЕДРА ІНФОРМАТИКИ І КОМП'ЮТЕРНИХ НАУК

Пояснювальна записка

до дипломної бакалаврської роботи

на тему: «Розробка інформаційної системи інтелектуального аналізу кредитних ризиків на основі алгоритмів градієнтного бустингу та методів SHAP»

Виконав: студент 4 курсу, групи 4КН

спеціальності 122 «Комп'ютерні науки»

Бадарла Т. П.

Керівник: Вишемирська С.В.

Рецензент: Огнєва О.Є

Хмельницький – 2026 р.

Факультет	**Інформаційних технологій та дизайну**
Кафедра	**Інформатики і комп'ютерних наук**
Рівень вищої освіти	**Перший (бакалаврський) рівень**
Галузь підготовки	**12 «Інформаційні технології»**
Освітньо – професійна програма	**Комп’ютерні науки**
Спеціальність	**122 «Комп'ютерні науки»**

ЗАТВЕРДЖУЮ

Завідувач кафедри ІКН

к.т.н., доцент

_____ С. В. Моїсеєнко

«____» _____ 2026 року

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Бадарлі Теодору Прасадовичу

(прізвище, ім’я, по батькові)

1. Тема роботи: «Розробка інформаційної системи інтелектуального аналізу кредитних ризиків на основі алгоритмів градієнтного бустингу та методів SHAP»

2. Керівник роботи Вишемирська С. В., кандидат технічних наук, доцент кафедри інформатики і комп’ютерних наук

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ХНТУ від «26» січня 2026 р. № № 133-с

3. Вихідні дані до роботи: Вихідними даними до проєкту є історичний набір фінансових показників позичальників «Give Me Some Credit» та

нормативно-методологічні вимоги концепції Explainable AI. Інформаційну систему реалізовано у вигляді розподіленої мікросервісної архітектури на основі мови Python (FastAPI) та веб-фреймворку Next.js. Прогностичне ядро побудовано за допомогою алгоритмів градієнтного бустингу LightGBM та методів інтерпретації моделей SHAP , а оркестрацію інфраструктури виконано в контейнерному середовищі Docker.

4. Зміст розрахунково-пояснювальної записки: Вступ; 1. Зрозумілість та інтерпретованість системи інтелектуального аналізу кредитних ризиків на основі машинного навчання; 2. Технології та інструменти для побудови моделей машинного навчання; 3. Практична реалізація інтелектуальної системи аналізу кредитних ризиків; 4. Розгортання мікросервісної архітектури; висновки; Список використаних джерел; Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): Рисунків –7, Формул – 2.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1.	Вишемирська С. В., к.т.н., доцент кафедри інформатики і комп'ютерних наук	26 січня 2026 року	23 травня 2026 року
2.	Вишемирська С. В., к.т.н., доцент кафедри інформатики і комп'ютерних наук	15 лютого 2026 року	25 травня 2026 року
3.	Вишемирська С. В., к.т.н., доцент кафедри інформатики і комп'ютерних наук	27 березня 2026 року	27 травня 2026 року
4.	Вишемирська С. В., к.т.н., доцент кафедри інформатики і комп'ютерних наук	12 квітня 2026 року	5 червня 2026 року

Дата видачі завдання 26.01.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми, формулювання мети та завдань, складання попереднього плану дипломної роботи.	24.02.2026 – 12.03.2026	Виконано
2	Пошук і опрацювання науково-технічної літератури з тематики кредитного скорингу, концепції Explainable AI (XAI), градієнтного бустингу та розподілених обчислень.	12.03.2026 – 24.03.2026	Виконано
3	Аналіз існуючих технічних підходів та аналітичних систем оцінювання ризиків. Визначення functional вимог до системи та проєктування її мікросервісної архітектури й API-шлюзу.	24.03.2026 – 30.03.2026	Виконано
4	Попередня обробка фінансового набору даних «Give Me Some Credit», розробка механізмів генерації синтетичних даних (CTGAN) та розгортання прогностичних моделей LightGBM із експлейнером SHAP.	30.03.2026 – 07.04.2026	Виконано
5	Програмна реалізація автономних серверних мікросервісів (FastAPI, Python) та розробка інтерактивного інтерфейсу користувача (Next.js, React, TypeScript).	08.04.2026 – 09.05.2026	Виконано
6	Комплекс інфраструктурного тестування, налаштування міжсервісної комунікації (Kafka, Redis), валідація моделей на предмет дрифту даних та оцінювання соціального капіталу.	10.05.2026 – 16.05.2026	Виконано
7	Систематизація отриманих результатів, написання та структурування теоретичних і практичних розділів дипломної роботи.	16.05.2026 – 29.05.2026	Виконано
8	Завершення технічного оформлення пояснювальної записки, підготовка презентаційних матеріалів та фінальна синхронізація публічного репозиторію GitHub до захисту.	30.05.2026 – 10.06.2026	

Студент _____ Бадарла Т.П.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Вишемирська С.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бадарла Т. П. Розробка інформаційної системи інтелектуального аналізу кредитних ризиків на основі алгоритмів градієнтного бустингу та методів SHAP. – Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю «Комп'ютерні науки» – ХНТУ, м. Хмельницький, 2026 рік.

Кваліфікаційну роботу присвячено розробці, комплексному об'єктно-орієнтованому проєктуванню та інфраструктурному розгортанню високонадійної мікросервісної інформаційної системи, призначеної для автоматизованого оцінювання фінансових ризиків і кредитного скорингу. Актуальність дослідження зумовлена стрімким розвитком штучного інтелекту та критичною потребою фінансового сектору у прозорих, інтерпретованих моделях машинного навчання, що функціонують в умовах жорстких регуляторних вимог фінансового аудиту.

Об'єктом дослідження є процес інтелектуального аналізу та оцінювання кредитних ризиків у сучасних фінансових інформаційних системах. Предметом дослідження виступають алгоритми градієнтного бустингу та математичні методи концепції Explainable AI (XAI) для інтерпретації результатів моделювання в межах клієнт-серверної архітектури.

У теоретичній частині роботи подолано проблему «чорного ящика» шляхом інтеграції прогностичного ансамблевого ядра LightGBM із математичним апаратом теорії ігор Shapley (методологія SHAP). Це дозволило детально декомпозувати логіку прийняття рішень алгоритмами ШІ й оцінити локальний та глобальний валідаційний внесок кожної індивідуальної характеристики клієнта з набору даних «Give Me Some Credit» у підсумкову ймовірність дефолту. Досліджено еластичність моделей за допомогою архітектури Human-in-the-Loop (HITL), каузальних моделей подвійного машинного навчання (DML), генеративно-змагальних мереж (CTGAN) для створення синтетичних даних та фінансової мовної моделі FinBERT.

Практична імплементація системи CredsCore виконана у вигляді розподіленої мікросервісної інфраструктури під керуванням центрального оркестратора, що об’єднує дев’ять автономних сервісів, запущених в ізольованому контейнерному середовищі Docker. Серверна логіка реалізована на базі фреймворку FastAPI (Python) з інтеграцією векторної бібліотеки Meta FAISS (алгоритми HNSW) для швидкого пошуку схожих кредитних історій та виявлення шахрайських кілець. Клієнтський інтерфейс розроблено з використанням бібліотек React 19 та фреймворку Next.js 16 (App Router) із суворою типізацією TypeScript та оптимізованим кешуванням даних через TanStack Query й Redis. Інтерфейс містить інтерактивну сторінку метрик із системою вкладок для моніторингу даних дрифту, агентського скорингу та візуалізації аналітики соціального капіталу клієнтів. вихідний код проєкту розміщено у публічному репозиторії GitHub.

Робота складається зі вступу, 4 розділів, загальних висновків, списку використаних джерел із 33 найменувань та 11 додатків.

Ключові слова: *кредитний скоринг, градієнтний бустинг, LightGBM, методи SHAP, Explainable AI, мікросервісна архітектура, FastAPI, Next.js, Docker, FAISS, каузальний аналіз, соціальний капітал.*

ABSTRACT

Badarla T. P. Development of an Intelligent Information System for Credit Risk Analysis Based on Gradient Boosting Algorithms and SHAP Methods. – Manuscript.

Bachelor’s thesis in Speciality 122 "Computer Science" – KNTU, Khmelnytskyi, 2026.

The qualification thesis is dedicated to the development, comprehensive object-oriented design, and infrastructural deployment of a highly reliable microservice-based information system designed for automated financial risk assessment and credit scoring. The relevance of the study is driven by the rapid development of artificial intelligence and the critical need of the financial sector for transparent, interpretable machine learning models operating under the strict regulatory requirements of financial auditing.

The object of the study is the process of intelligent analysis and credit risk assessment in modern financial information systems. The subject of the study includes gradient boosting algorithms and mathematical methods of the Explainable AI (XAI) concept for interpreting modeling results within a client-server architecture.

In the theoretical part of the work, the "black box" problem is overcome by integrating the predictive ensemble kernel LightGBM with the mathematical apparatus of Shapley game theory (SHAP methodology). This allowed for a detailed decomposition of the decision-making logic of AI algorithms and an evaluation of the local and global validation contribution of each individual customer characteristic from the "Give Me Some Credit" dataset into the final default probability. The resilience of the models was investigated using the Human-in-the-Loop (HITL) architecture, causal Double Machine Learning (DML) models, generative adversarial networks (CTGAN) for synthetic data generation, and the financial language model FinBERT.

The practical implementation of the CredsCore system is executed as a distributed microservice infrastructure under the control of a central orchestrator, combining nine autonomous services launched in an isolated Docker container

environment. The server-side logic is implemented based on the FastAPI framework (Python) with the integration of the Meta FAISS vector library (HNSW algorithms) for high-speed similarity search of credit histories and the detection of fraud rings. The user interface is developed using React 19 libraries and the Next.js 16 framework (App Router) with strict TypeScript typing and optimized data caching via TanStack Query and Redis. The interface includes an interactive metrics page with a tab system for monitoring data drift, agentic scoring, and visualization of customers' social capital analytics. The project's source code is hosted in a public GitHub repository.

The thesis consists of an introduction, 4 chapters, general conclusions, a reference list of 33 items, and 11 appendices.

Keywords: *credit scoring, gradient boosting, LightGBM, SHAP methods, Explainable AI, microservice architecture, FastAPI, Next.js, Docker, FAISS, causal inference, social capital.*

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ ТА ТЕРМІНІВ

ЕК – Екзаменаційна комісія.

НБУ – Національний банк України.

ШІ – Штучний інтелект.

API – Application Programming Interface (прикладний програмний інтерфейс).

BNN – Bayesian Neural Network (Байєсівська нейронна мережа).

CORS – Cross-Origin Resource Sharing (спільне використання ресурсів різних джерел).

CSS – Cascading Style Sheets (каскадні таблиці стилів).

CTGAN – Conditional Tabular Generative Adversarial Network (умовна таблична генеративно-суперницька мережа).

CUDA – Compute Unified Device Architecture (програмно-апаратна архітектура паралельних обчислень від компанії NVIDIA).

D3.js – Data-Driven Documents (JavaScript-бібліотека для створення інтерактивної візуалізації даних у веб-браузерах).

DML – Double Machine Learning (подвійне машинне навчання).

ELBO – Evidence Lower Bound (нижня оцінка доказовості у байєсівському висновку).

FAISS – Facebook AI Similarity Search (векторна бібліотека для швидкого пошуку схожості та кластеризації високомірних векторів).

GAN – Generative Adversarial Networks (генеративно-суперницькі нейронні мережі).

GBDT – Gradient Boosting Decision Tree (градієнтний бустинг на основі дерев рішень).

GDPR – General Data Protection Regulation (Загальний регламент про захист даних Європейського Союзу).

GPU – Graphics Processing Unit (графічний процесор).

HITL – Human-in-the-Loop (концепція залучення людини в контур обчислень / інтерактивне машинне навчання).

HNSW – Hierarchical Navigable Small World (ієрархічний навігаційний граф малого світу для наближеного пошуку найближчих сусідів).

HPC – High-Performance Computing (високопродуктивні обчислення).

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту).

HTTPS – HyperText Transfer Protocol Secure (захищений протокол передачі гіпертексту з підтримкою шифрування).

IDE – Integrated Development Environment (інтегроване середовище розробки програмного забезпечення).

IPW – Inverse Probability Weighting (зважування за оберненою ймовірністю у каузальному аналізі).

JSON – JavaScript Object Notation (текстовий формат обміну даними між сервісами).

LightGBM – Light Gradient Boosting Machine (оптимізований високопродуктивний фреймворк градієнтного бустингу).

ML – Machine Learning (машинне навчання).

NLP – Natural Language Processing (обробка природної мови).

NMSLIB – Non-Metric Space Library (бібліотека неметричних просторів для пошуку найближчих сусідів).

OFAC – Office of Foreign Assets Control (Управління з контролю за іноземними активами Міністерства фінансів США).

ROCm – Radeon Open Compute (відкритий програмний стек компанії AMD для апаратного прискорення обчислень на GPU).

SCSS – Sassy Cascading Style Sheets (діалектна мова стилів, що компілюється в CSS).

SDK – Software Development Kit (набір інструментів та бібліотек для розробки програмного забезпечення).

SHAP – SHapley Additive exPlanations (метод локальних та глобальних адитивних пояснень на основі теорії ігор Shapley).

SNN – Shared Nearest Neighbors (метод пошуку подібності на основі спільних найближчих сусідів).

SSN – Social Security Number (номер соціального страхування / ідентифікатор для верифікації особи).

UI – User Interface (інтерфейс користувача).

URL – Uniform Resource Locator (єдиний вказівник цифрового ресурсу).

WCAG – Web Content Accessibility Guidelines (міжнародні настанови з доступності веб-вмісту для користувачів).

XAI – Explainable Artificial Intelligence (концепція штучного інтелекту, що піддається поясненню / прозорого машинного навчання).

ВСТУП

Актуальність проблеми. З нещодавнім стрімким розвитком штучного інтелекту, інтерес поступово зростає в таких сферах як машинне навчання, фінанси та дата-аналітика. Зі збільшенням можливостей складність розуміння моделей машинного навчання також підвищується, й стає важче передбачити можливий результат. Особливо у фінансах, тому що ця сфера потребує достатньої точності у розрахунках, а також здатності проаналізувати вхідний кредитний ризик, враховуючи мінімальну машинну неточність.

Дата-алгоритми (випадковий ліс, градієнтний бустинг), а також порівняння натренованих моделей дозволять нам зрозуміти як моделі аналізу кредитоспроможності працюють всередині, які алгоритми вони використовують та чи дійсно вони надійні. Проблема має зв’язок з веб-розробкою, тому що система аналізу кредитоспроможності потребує доступу до Інтернет-підключення. А також наявності клієнт-серверної архітектури, що здобувається за допомогою API-запитів. Відсутність цього зв’язку може стати причиною поганого з’єднання або порожніх відповідей з серверу.

Ця причина має схожість з проблемами машинного навчання, де алгоритми розрахунків застосовуються до необробленого датасету з наявністю викидів. Це також стосується неточності відтворення отриманої інформації використовуючи похибки в процесі створення фронтенд-частини, а також у процесі побудови інтерактивних інформаційних панелей та схожих візуальних представлень показників системи інтелектуального аналізу кредитних ризиків. Відсутність зрозумілої інформаційної панелі може спричинити окремий тип проблеми, такої як нездатність забезпечити користувача візуальними висновками їх кредитоспроможності. Ці чотири проблеми є основною актуальністю роботи.

Виходячи із актуальності дослідження можемо сформулювати **мету роботи** – розробка високонадійної, масштабованої інформаційної системи інтелектуального аналізу кредитних ризиків на основі мікросервісної архітектури, що забезпечує високу точність прогнозування за допомогою

алгоритмів градієнтного бустингу та прозорість (інтерпретованість) прийняття рішень за допомогою методів XAI (зокрема SHAP).

Об’єкт дослідження. Процес інтелектуального аналізу та оцінювання кредитних ризиків у фінансових інформаційних системах.

Предмет дослідження. Алгоритми градієнтного бустингу та методи SHAP для моделювання й інтерпретації результатів кредитного скорингу, а також методи їх програмної реалізації в межах клієнт-серверної архітектури.

Для досягнення поставленої мети необхідно вирішити такі **задачі**:

1. Дослідити проблему «чорного ящика» в моделях машинного навчання для кредитного скорингу та проаналізувати сучасні підходи до забезпечення інтерпретованості (XAI).

2. Обґрунтувати вибір алгоритму LightGBM, методів SHAP, а також засобів апаратного прискорення (CUDA/ROCm) та векторного пошуку (FAISS) для оптимізації обчислень.

3. Дослідити можливість застосування допоміжних інтелектуальних методів (GAN для генерації синтетичних даних, каузального аналізу, мовних моделей FinBERT) для збагачення аналітики ризиків.

4. Спроектувати та реалізувати клієнтську частину системи з використанням Next.js App Router, TypeScript та забезпечити ефективну клієнт-серверну комунікацію через API.

5. Розробити та розгорнути мікросервісну архітектуру системи, реалізувавши спеціалізовані сервіси (аналізу ризиків, виявлення шахрайства, каузального аналізу, оцінювання соціального капіталу та генерації даних).

Методи дослідження. Для розв'язання поставлених задач у роботі використано:

– *методи машинного та глибокого навчання* (алгоритми градієнтного бустингу, ансамблеві моделі LightGBM) – для класифікації та прогнозування дефолту;

– *методи концепції штучного інтелекту, що піддається поясненню (XAI)*, зокрема теорію ігор Shapley (методи SHAP) – для інтерпретації моделей;

– *методи генеративно-змагальних мереж (GAN)* – для створення синтетичних вибірок даних;

– *методи обробки природної мови (NLP)*, зокрема модель FinBERT – для аналізу темпоральних та текстових фінансових показників;

– *методи теорії графів та векторної подібності* (метрики SNN, бібліотека FAISS) – для валідації кредитних історій;

– *методи системного аналізу та програмної інженерії* (мікросервісна архітектура, контейнеризація, клієнт-серверна модель) – для проектування та реалізації інформаційної системи.

Наукова (технологічна) новизна одержаних результатів полягає у комплексному поєднанні методів інтерпретованого машинного навчання (SHAP) та мультиагентного підходу в межах єдиної мікросервісної платформи оцінювання ризиків. Запропоновано архітектурне рішення, яке інтегрує не лише класичні скорингові дані, а й каузальні зв'язки, текстову семантику (FinBERT) та оцінку соціального капіталу користувача, що дозволяє підвищити об'єктивність кредитного аналізу за збереження повної прозорості алгоритму.

Практичне значення одержаних результатів полягає у створенні готової до розгортання та масштабування інформаційної системи інтелектуального аналізу кредитних ризиків. Розроблена архітектура дашборду (Next.js) та мікросервісів дозволяє фінансовим установам знизити рівень непрацюючих кредитів (NPL), автоматизувати процес прийняття рішень та забезпечити відповідність регуляторним вимогам щодо прозорості AI-моделей.

Практична імплементація розробленої інтелектуальної системи виконана у вигляді відкритого програмного забезпечення, вихідний код та архітектурна структура якого розміщені у публічному репозиторії на платформі GitHub за посиланням: https://github.com/Hakkarian/CredsCore.

Апробація результатів дослідження. Основні положення та результати роботи доповідалися на VIII Всеукраїнській науково-практичній інтернет-конференції студентів, аспірантів та молодих вчених за тематикою «Сучасні комп’ютерні системи та мережі в управлінні», що відбулася 24 листопада

2025 р., м. Херсон, м. Хмельницький (Badarla T., Vyshemyrska S. A Comprehensive Description for Smart Home Virtual Projecting. **Сучасні комп’ютерні системи та технології**: матеріали VIII Всеукр. наук.-практ. інтернет-конф. студентів, аспірантів та молодих вчених за тематикою «Сучасні комп’ютерні системи та мережі в управлінні» (24 листопада 2025 р., м. Херсон, м. Хмельницький) / за ред. А. А. Григорової. – Херсон: Книжкове видавництво ФОП Вишемирський В. С., 2025. – С. 57-59.).

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, 4 розділів, висновків, списку використаних джерел із 33 найменувань та 13 додатків. Повний обсяг роботи становить 112 сторінок комп'ютерного тексту, включаючи 7 рисунків.

ЗМІСТ

ВСТУП 2

РОЗДІЛ 1. ЗРОЗУМІЛІСТЬ ТА ІНТЕРПРЕТОВАНІСТЬ СИСТЕМИ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ КРЕДИТНИХ РИЗИКІВ НА ОСНОВІ МАШИННОГО НАВЧАННЯ 18

1. 1. Вплив самовдосконалення людини на розвиток штучного інтелекту: архітектура Human-in-the-Loop 18

1. 2. Проблема «чорного ящику» в машинному навчанні та регуляторні вимоги 20

1. 3. Аналіз наукових робіт з кредитного скорингу та XAI 22

1. 4. Соціальні ризики та відповідальність за технічну імплементацію 24

1. 5. Визначення, ознаки кредитного скорингу та інтерпретованості в машинному навчанні 26

1. 6. Точність моделей: LightGBM, SHAP та альтернативи 28

1. 7. Надбудова LightGBM для оптимізації фінансових операцій: ROCm та CUDA 29

1. 8. Роль SNN у валідації кредитної історії за пошуком схожості між користувацькими сценаріями 32

1. 9. Векторна бібліотека FAISS та її роль у системі аналізу кредитних ризиків 33

1. 10. Кращі проектні практики: модульність, масштабованість, безпека 35

РОЗДІЛ 2. ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ ДЛЯ ПОБУДОВИ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ 37

2. 1. Використання GAN у генерації синтетичних клієнтських даних 37

2. 2. Каузальні моделі, подвійне машинне навчання та п’ятикратне перехресне тестування 38

2. 3. CreditXAI як кредитна система колаборації агентів та субагентів 39

2. 4. Контрастне навчання на основі комбінованих ознак 41

2. 5. Байєсівське машинне навчання та комбінований кредитний інтелект 42

2. 6. LendNova, мовна модель FinBERT та темпоральні вектори 44

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ АНАЛІЗУ КРЕДИТНИХ РИЗИКІВ 46

3. 1. Визначення та побудова архітектури 46

3. 2. Файлова структура клієнтської частини 48

3. 3. Next.js App Router та конфігурація маршрутів. 49
3. 4. Клієнт-сервісна комунікація за допомогою API 50
3. 5. Typescript і проектна типізація. 51
РОЗДІЛ 4. РОЗГОРТАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ 56
4. 1. Диригент мікросервісної архітектури. 56
4. 2. Сервіс аналізу кредитних ризиків 57
4. 3. Розробка сервісу виявлення шахрайства 60
4. 4. Сервіс причинно-наслідкових зв’язків 62
4. 5. Створення сервісу соціального капіталу 65
4. 6. Сервіс доповненого кредитного аналізу 70
4. 7. Сервіс прийняття рішень та організації процесу між сервісами 76
4. 8. Сервіс збагачення синтетичними даними 83
4. 9. Загальний сервіс, пакети та інфраструктура 92
ВИСНОВКИ 100
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 103
Додатки 108

РОЗДІЛ 1. ЗРОЗУМІЛІСТЬ ТА ІНТЕРПРЕТОВАНІСТЬ СИСТЕМИ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ КРЕДИТНИХ РИЗИКІВ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

1. 1. Вплив самовдосконалення людини на розвиток штучного інтелекту: архітектура Human-in-the-Loop

Оскільки машинне навчання є основою штучного інтелекту, розвиток машинного навчання є причиною часткової відсутності пошуків самовдосконалення людини в сучасному світі. Штучний інтелект на сьогоднішній день здатний значно полегшити роботу в багатьох сферах, що стосуються не лише роботи з даними, а й поведінки в ситуаціях, де точність не має залежати від людського фактору. Під роботою мається на увазі діяльність, що спрямована на розвиток конкретної галузі з метою її збагачення цифровими та фізичними даними.

Незважаючи на втручання людини, робота може виконуватись цілком автоматично. Автоматизація роботи залежить від того, наскільки алгоритм навчання моделі, що представляє собою окремий аспект машинного навчання, точно виконує пункти, за якими рухається. Цими пунктами є як тестування на оброблених та інтерпретованих датасетах, так і додавання тригерів, за якими будуть виконуватись подальші дії. Активація цих тригерів нагадує деревоподібний розвиток з коренів до листків, де коренями є початкові умови, а листки – досягнутий результат за похідними від цих умов. Деревоподібний перехід від однієї умови до іншої є достатньо зрозумілим, адже дозволяє структурувати інформацію.

Коли йдеться про обсяги даних, які людина має опрацювати за кілька годин, натренована модель виконає подібне завдання за хвилини. Проте машинний результат може відрізнятись від людського за кривою ефективності/витраченого часу. Згідно зі статею, поточні «моделі мають майже 100% успіх при виконанні завдань, на які людина витрачає до 4-х хвилин. Але їх ефективність спадає до 10% коли йдеться про роботу з тривалістю до 4 годин» [1]. Враховуючи таку

ефективність, це спонукає людину застосовувати більш точні метрики, та планувати завдання покроково. Це нагадує ділення технічного проекту на віхи, підзадачі та баги. Відсутність такого підходу підвищує шанс появи «галюцинацій» у відповідях. Чим більший масштаб майбутнє завдання займатиме, тим вищий шанс похибок. Внаслідок чого, людина самовдосконалюється, бо переконуючись в неправдивості отриманої інформації, шукає більш надійні джерела. В процесі цих пошуків вона навчається сама, а отже, навчається і модель. Адже у процесі отримання уточнень датасет поступово покращується.

У статті на Amazon, описана вище архітектура взаємовідносин моделі з людиною називається Human-in-the-Loop (тут і далі - HITL) [2]. Мета цієї філософії – "знаходження та зупинка галюцинацій перед тим, як вони досягнуть користувачів, а також покриття критичних ситуацій, де модель не може знайти істинну відповідь". HITL покладається на чотири методи: схвалення відповіді; перевірка та редагування людиною; ескалація до людського втручання у реальному часі; відгук самого користувача на відповідь. У деревоподібній структурі розглянуто умови, за якими ці методи використовуються. До прикладу, у ситуаціях коли спілкування на заплутані теми з користувачем виходять за межі компетенції або користувач явно просить допомоги людини, відбувається ескалація до реального втручання експерта.

Згідно з деревом рішень HITL, щоб досягти схвалення модельного результату, необхідно відтворити наступний алгоритм питань-відповідей:

«Чи потрібне реальне втручання?

Ні. Чи є завдання двомірним рішенням?

Так. Чи є рівень ризику високим (затвердження кредиту)?

Так».

Детально аналізуючи наведену схему (див. рис. 1), рішення розгалужуються вже на етапі питання «*чи є завдання двомірним рішенням?*», де «*так*» призводить лише до можливої ескалації чи користувацького відгуку, а «*ні*» «*відтинає*» можливість досягти ескалації.

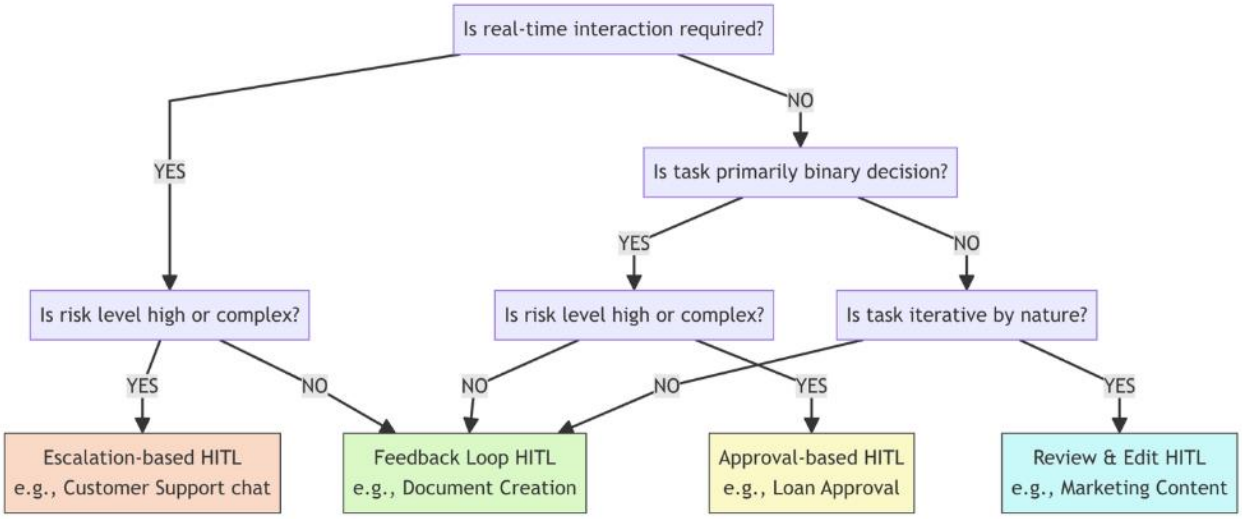


Рисунок 1. Дерево рішень для вибору патерну HITL (людина в контурі)

Архітектура HITL дозволяє залучати людину в цикл послідовного навчання моделі за допомогою отримання користувацьких відгуків на модельні відповіді, та з можливим втручанням людських експертів. Результатом такого підходу датасет з нижчою вірогідністю «галюцинацій», натренований і готовий до тестування. Втім, зі зростанням точності, підвищується й рівень складності опанування моделі.

1. 2. Проблема «чорного ящику» в машинному навчанні та регуляторні вимоги.

Для забезпечення регуляторних вимог необхідна точність та зрозумілість інформації, проте ці поняття є протилежними, адже точність отриманої інформації потребує розгорнутого алгоритму навчання та тестування моделі у відношенні 4:5, тобто чотири датасети для тренування, і п’ятий невідомий для навчання, що підходить для досягнення крос-валідації. Крім того, рівень якості моделі залежить від стабілізації параметрів та запобігання перенавчання. Це інтерпретує представлення складності та незрозумілості накопичених налаштувань як «об’єкт бога», тобто об’єкт який виконує усі дії, інакше кажучи, «чорний ящик». Його можна порівняти з закритим вихідним кодом з відсутністю прозорості.

Як вважає штатний редактор M. Kosinski [3], «непрозорість обчислень досягається завдяки бажанню AI-розробників – власник знає все про внутрішню роботу моделі, проте спеціально уникає подробиць заради збереження власних прав. У статті автор не підтверджує визначення терміну «чорний ящик» як аналогу непрозорості закритого вихідного коду, а скоріше самого коду. Відкритий код також має приховані частини, незрозумілі розробникам, оскільки торкаються «глибокого навчання» (тут і далі – deep learning)». На нашу думку, deep learning є поняттям у нейронній мережі, яке означає тренування моделі за допомогою декількох рівнів нейронів, кожний з яких інтерпретує конкретну частину отриманого масиву інформації, а результат відправляє кількості нейронів, що дорівнює сумі нейронів у поточному рівні.

За думкою автора [3], «чорний ящик» впливає на безпеку, відкриває вразливості, та є критично небезпечним у сфері медицини. Результати генеративного AI можуть бути хоч і точніші за прогнозовані моделі, проте їх результат може спиратись на інші параметри. Автор наводить приклад «ефекту розумного Ганса», де кінь Ганс вміє рахувати шляхом обчислення кількості тупотінь копитом. На практиці, кінь аналізує рухи тіла власника. Результат може бути ідентичний, але досягнутий іншим шляхом.

Щоб мінімізувати наслідки «чорного ящику» [4] Управління США з контролю за іноземними активами (OFAC) закликає дотримуватись наступних регуляторних вимог: перевіряти налаштування на можливі помилки, датасети на цілісність даних, і автоматизовані процеси на прозорість кожного наступного етапу. Для того, щоб модель зберігала свою зрозумілість, необхідно документувати її дизайн. Розуміти, на якій інформації ця модель тренується та яка логіка знаходиться всередині.

Автор підкреслює роль людини: яка б модель не була налаштована, завжди залишатимуться ситуації, де інформації недостатньо, і не завжди зрозуміло в чому полягає ризик. На допомогу приходить гібридний метод – штучний інтелект опрацьовує повторювані завдання низької складності, тоді як людина знаходить важкодоступні похибки. З точки зору регуляторних вимог такий метод є набагато

безпечніший, ніж повністю автоматизований. Інформація завжди має бути релевантною, це також стосується й самих вимог, адже з кожним роком еволюціонують не лише технології, але й інструменти для їх зламу.

Крім OFAC існують також Управління Фінансової Поведінки Великобританії, Грошово-кредитне управління Сінгапуру, а також Європейське банківське управління. В Україні Міністерство Цифрової Трансформації та Національний Банк України також просувають вимоги про прозорість та відповідальність моделей, заснованих на релевантній інформації. За словами Є. Бухальського, «ШІ може аналізувати отриману інформацію, але остаточне рішення залишається за співробітником банку» [5].

Регуляторні вимоги зазначають, що людина є головним валідатором [4] в автоматизованому процесі машинного навчання. Проблема «чорного ящику» є присутня також і системі аналізу кредитних ризиків, а отже слід опрацювати наукові роботи за цією темою.

1. 3. Аналіз наукових робіт з кредитного скорингу та XAI

На нашу думку, XAI є системою регуляторних вимог, за якими моделі машинного навчання повинні бути достатньо зрозумілими, менш вразливими до помилок та хакерських атак, більш спроможними перейти крізь надати користувачу перелік пояснень стосовно отриманого результату. Оскільки система інтелектуального аналізу кредитних ризиків стосується фінансової сфери, правила XAI мають високий пріоритет, і дозволяють клієнту зрозуміти в чому причина відхилення кредиту чи схвалення кредитної спроможності.

Як стверджує A. Rao, «традиційні моделі кредитного скорингу, зокрема логістичної регресії, домінують у сфері фінансових операцій через властиву їм інтерпретованість та узгодження з регуляторними стандартами XAI. Чудова прогностична спроможність моделей машинного навчання походить з їх здатності автоматично реагувати на складні шаблони у заплутаних датасетах, не вимагаючи ручної розробки функцій» [6].

Однак, у цьому випадку, автоматизм може впливати й негативно на складні набори інформації, оскільки не всі змінні враховуються, а можуть і бути видозміненими в процесі обчислення. «SHAP обчислює атрибути ознак, які є граничним внеском кожної змінної у відхилення індивідуальних прогнозів від базових значень, і внески ці підсумовуються до різниці між значеннями екземплярів та попередніми значеннями у всьому датасету» [6].

Можна пояснити це так: ми маємо задане значення в діапазоні X, а також Y. X є загальним значенням, а Y індивідуальним. Значення Y може бути вище або нижче X, і SHAP допомагає зрозуміти, *чому* Y відрізняється від X. SHAP аналізує кожну ознаку Y, та вирішує, які з них найбільше повпливали на різницю між X. SHAP полягає у застосуванні формули (1):

$$\phi_i(f,x)=\frac{1}{|N|}\sum_{S\subseteq N\setminus\{i\}}\frac{\left[f\left(x_{S\cup\{i\}}\right)-f\left(x_S\right)\right]}{\binom{|N|-1}{|S|}} \tag{1}$$

Результатом виконання формули (1) є обчислення середньої всіх варіантів підмножин разом з заданою індивідуальною ознакою, знаходження та сума всіх різниць, і ділення цієї суми на кількість ознак у всьому датасеті. Завдяки цій формулі SHAP знаходить середнє значення ознаки, як наприклад розташування, поштовий індекс, вік тощо. Таким чином, різницю можна пояснити, спираючись на відхилення індивідуальних ознак на фоні загальних. Bahlool R. з посиланням на Valdrighi G. підтверджує існування індивідуальних та групових ознак об’єктивності. Групова ознака полягає у генералізації кредитних рішень між групами, що охарактеризовані такими атрибутами як гендерна ознака, етнічна приналежність та релігія [7].

Індивідуальною ознакою об’єктивності є рівне відношення до окремих людей. Прикладом цього є рівне ставлення до двох кандидатів з однаковими технічними навичками, але з різною біографією [8]. Відсутність рівності між груповими чи індивідуальними ознаками об’єктивності є упередженість. Вона може впливати на модель в залежності від наявності домінуючих ознак у

датасеті. Через воєнний стан в Україні, за програмою пільгової іпотеки єОселя рівень відмов залишається високий (лише 10-30 відсотків респондентів отримують доступ до іпотеки [9]), що є часткою людської упередженості. У США, через дискримінацію та через історичну упередженість, було відмовлено у 1,3 мільйонах іпотечних кредитів [7]. Точність моделей не є завжди запорукою якості, оскільки вони точно обчислювати інформацію, але з упередженої точки зору, тобто мати приналежність до категорії соціального ризику.

1. 4. Соціальні ризики та відповідальність за технічну імплементацію

Упередженість залишається соціальною проблемою під час розробки інтелектуальної системи кредитних ризиків. Моделі машинного навчання, особливо натреновані, складаються з великої кількості історичної інформації, яка охоплює й ті часи, де відмова нацменшинам у надання послуг, та обмеження лише використанням білими було нормою. Проблема «чорного ящику» блокує розв'язання упередженості, оскільки деталі машинної моделі або важко пояснити, або контролюються власниками [10]. Внаслідок цього людина отримує відмову, навіть якщо індивідуальні ознаки її кредитної історії в сумі не перевищували 0.1%.

Ця упередженість стосується не лише по відношенню людина до людини, а й машинна упередженість. У базі даних, людська ідентичність скорочується до «двомірного портрету» [11, С. 42]. Абстракція людей у якості чисел, які модель машинного навчання може інтерпретувати. Таким чином, автор порушує проблему унікальності людини, де на сьогоднішній день людина має підлаштовуватись під стандарти машини, а не навпаки. Слід також сказати, що модель машинного навчання не лише деформує людську ідентичність, а й забирає контроль у питанні персональної кредитної історії. Внаслідок цього, а також ігноруванням машиною особливостей людини, клієнт не здатний зміни власну ситуацію, отримуючи лише кінцевий результат.

Соціальним ризиком є також розташування – в залежності від zip-коду чи поштовому індексу, ці ознаки можуть значно впливати на відмову в кредиті.

Vilarino R., досліджуючи упередженість бразильської моделі стосовно поштових індексів, показує (рис. 2) як поштовий індекс CEP-3 проявляє дискримінацію стосовно інших індексів, в той час як значення CEP-3 є порівняно низький.

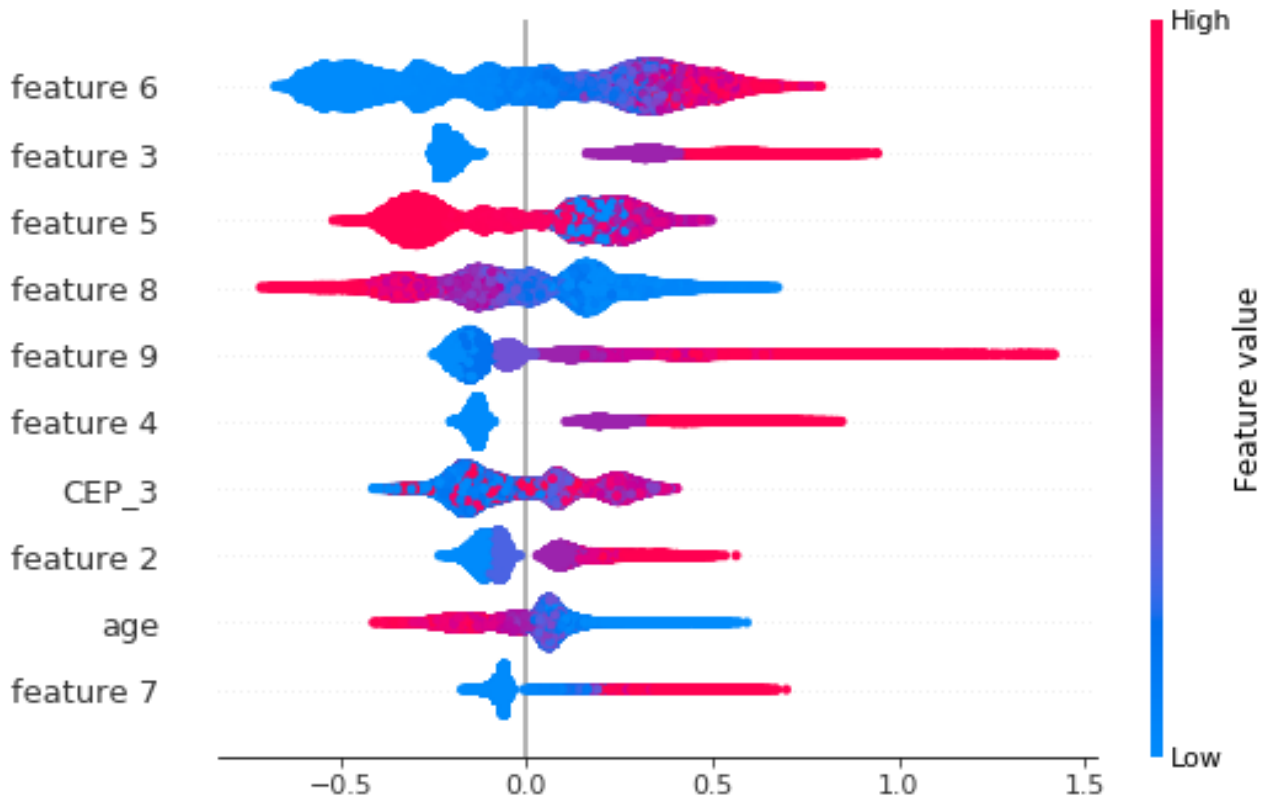


Рисунок 2. Значення SHAP (вплив на результат моделі)

Значення спостерігається на вісі X, де синій колір означає зменшення домінантності ознаки, а червоний – збільшення. У додатку А3 зображено, що CEP-3 зберігає стабільний стан, у той час як в А4 ознака вказує на зростаючий позитивний результат. Таким чином, однакова ознака поштового індексу дає різний результат в залежності від розташування [11]. Що стосується відповідальності за технічну імплементацію, модель повинна мати зрозумілі налаштування, ознака «білого ящику» ще на старті проекту для запобігання уникнення документованості.

За це відповідальний власник, адже розробник діє за його вказівками, та здатний як розширити потужність моделі, так і закрити доступ до перегляду, зберегти приватність. Як стверджує Schmidt J-H., «процеси технічної

імплементації та результуюча відповідальність потребують різних підходів до комунікації та керівництва. Порівняно з прогресуючою відповідальністю, результуюча є уособленням розуміння AI-розробників зв'язку між їх поточними завданнями, та майбутніми наслідками. Також візуалізація аргументів відповідальності у вигляді UI-дизайну, доданих в IDE, підвищують рівень відповідальності AI-працівників» [12, С. 130-131]. IDE (Integrated Development Environment) є середовищем розробки, який буде мати основну роль під час практичної частини розробки інтелектуальної системи кредитних ризиків на основі машинного навчання.

1. 5. Визначення, ознаки кредитного скорингу та інтерпретованості в машинному навчанні

Кредитний скоринг є системою інтелектуального аналізу кредитних ризиків, спрямовану на вивчення кредитної історії, наданої клієнтом, та відправка результату схвалення або заборони. Як стверджує банк Barklays, «це система, що використовується багатьма фінансовими компаніями для проведення рішень стосовно клієнтських додатків для отримання позики. Це також включає в себе обчислення рейтингу кредитоспроможності, з інформації яку клієнт надав додатку, включно з інформацією про вік, роботу та фінансові зобов'язання. Перевірка також налагоджена з кредитними агенціями щоб побачити чи в них є додаткова інформація, яка може вплинути на кінцеве рішення» [13]. Зв'язок з цими сторонніми агенціями означає можливість отримати додаткові фінансові ознаки з рівнем домінантності, що є ключовою у агенції що надає інформацію.

Оскільки додаткову фінансову інформацію можуть надавати декілька агенцій, в порівнянні їх звітів упередженості не уникнути. За статтею ICO (Information Commissioner's Office), «більшість інформації, якою володіють агенції відноситься до якості управління власного кредиту та сервісні/комунальні рахунки. Ця інформація також включає колишні адреси проживання та публічні згадування у електоральних підрахунках голосів. Агенції використовують ці дані для боротьби з «відмиттям грошей», щоб відслідкувати можливу корупцію та для

передачі до уряду» [14]. Присутність додаткових агенцій важлива в тому випадку, якщо повнота клієнтських даних лише часткова або потребує стороннього підтвердження. Отримуючи клієнтську історію, модель має відповідати відповідно, тобто бути інтерпретованою. На думку автора, «щоб модель дійсно була інтерпретованою, вона має бути надійною – усі ідентичні запити мають мати стабільні пояснення.

Нестабільні відповіді не лише ставлять під загрозу документованість моделі, але й піддають фінансові установи репутаційним ризикам та можуть блокувати клієнтам доступ до оновлення своїх кредитних профілів» [15]. Щоб модель була стабільною, вона має відповідати без «галюцинацій». Під цим терміном йдеться про конкретні, невигадані відповіді. Складовою правильного використання моделі є відсутність зловживання контекстом. У кожної моделі є контекст, інакше кажучи збереження даних у пам’яті в одному чаті. Коли розмова між людиною та моделею триває довго, пам’ять заповнюється.

Щоб звільнити слот пам’яті для нових даних, модель «забуває» першу отриману інформацію для збереження останньої. Якщо першим коментарем були цінні вказівки для всього чату, зникнення їх з пам’яті може призвести до непередбачуваних відповідей. Для боротьби з такою поведінкою слід переходити до нового чату, коли контекст у старому чаті заповнюється повністю або близький до заповнення. Для збереження інтерпретованості також слід користуватись глобальним контекстом, який застосовується між всіма чатами.

Прикладом моделі можна запропонувати Sonnet 4.6 від Claude AI [16]. При налаштуванні глобального контексту, модель здатна зберігати свою інтерпретованість під час комплексних запитів, що потребують знання історії переписки. Кожна модель, так чи інакше, але має свій ліміт контексту, на який необхідно спиратись для отримання якісного контенту та збереження ознак інтерпретованості. Ми вважаємо, що інтерпретованість еквівалентна не лише стабільності моделі, але й її точності.

1. 6. Точність моделей: LightGBM, SHAP та альтернативи

Ми достатньо розглянули спосіб документування моделі завдяки формулі SHAP (див. додаток А2), тепер необхідно зрозуміти, як це впливає на точність. Алгоритми машинного навчання на основі дерев, такі як Extra Trees (ET), Random Forest (RF), Gradient Boosting Machine (GBM) та XGBoost (XGB) широко використовуються у відкритті ліків через їх універсальність. Однак, ці алгоритми потерпають від неправильної класифікації та недостатньої інтерпретованості, що обмежують їх дієздатність на практиці. Крім з'ясування ознак, що впливають на відповідь моделі, автор стверджує, що SHAP також можна адаптувати до покращення її дієздатності. Для кожної молекулярної ознаки, що модель правильно прогнозувала, обчислюється глобальний поріг та межа згідно окремого кластеру. Глобальним порогом є список високих значень ознак (тут і далі персентилів) (80%, 85%), отриманих від правильно прогнозованих ознак моделі.

Для обчислення кластерних персентилів, той же процес повторюється, але в межах окремих кластерів. Далі, згідно правильно прогнозованих ознак, для кожної ознаки їх значення порівнюється зі значення діапазону ознаки протилежного класу. Якщо значення в діапазоні, вона є протилежною. SHAP підраховує кожне з цих протилежних значень, та порівнює з персентилями правильного прогнозу. Якщо протилежні значення ознак молекули вищі ніж персентилі, така молекула буде вважатись некласифікованою [17, С. 9]. SHAP у цьому випадку використовується як аналізатор тригерів протилежних значень ознак молекули, дозволяючи зрозуміти на основі масиву протилежно класифікованих ознак, чи є молекула класифікованою. Погляд на SHAP у контексті аналізу, підбір персентилів та реакції на протилежні ознаки збільшує точність моделі.

В градієнтного бустингу, який згадувався на початку цього пункту, для збільшення точності використовується LightGBM алгоритм. За автором, «LightGBM є найбільш ефективним алгоритмом, з найменшими або відсутністю

втрат під час прогнозу. Серед імовірнісних алгоритмів XGBoostLSS є не лише ефективною в обчисленнях, але й точною» у формуванні відповідей. (розповісти про алгоритм LightGBM). В докумнетації до цього інструменту сказано, що «LightGBM використовує розщеплення на основі гістограми та ріст дерев leaf-wise для отримання швидкої зближення наборів даних з багатьма категоріальними змінними (наприклад тип зайнятості та статус верифікації). LightGBM підтримує низький обсяг пам'яті та якісно обробляє розрідженість даних, що робить LightGBM добре пристосованим до кредитного ризику та оцінки робочого навантаження. Завдяки оптимізованому обчисленню на гістограмі, прискорення GPU може бути в 2, а то й 5 разів більше».

Втім, автор застерігає від перенавчання моделі, якщо LightGBM використовується без параметру min_data_in_leaf [18]. Ми вважаємо, що цей параметр відповідає за мінімальну кількість інформації в листках дерева, і якщо не задати цю кількість явно, наприклад 36 екземплярів у одному листі, алгоритм може створити лише 5 чи 4, створюючи випадковий шум, який може серйозно впливати результат, якщо листів за таким шумом десятки тисяч. Оскільки LightGBM значно прискорює GPU, інформація в наборах даних обробляється паралельно, а отже з'являється бажання опрацювати більше даних. LightGBM є особливо потужним разом з набором інструментів ROCM від AMD Radeon у фінансовій сфері, оскільки використовується багато категоріальних ознак. ROCM, як додаток до LightGBM, ще більше прискорює GPU через оптимізовані ядра та кращу обробку пам'яті.

1. 7. Надбудова LightGBM для оптимізації фінансових операцій: ROCm та CUDA

«ROCm є програмним стеком, який складається з відкритого коду програмного забезпечення, що забезпечує інструментами для програмування графічного процесору (GPU) від ядер низького рівня до кінцевого користувацького пристрою високого рівня» [19]. Низький рівень ядер означає рівень коду, який є ближчий до зрозуміння машинам, ніж людині. Чим

зрозуміліша робота механізму для людини, тим вищий рівень. На рисунку 3 зображено, що ROCm покриває час виконання, компілятори, інструменти та бібліотеки, де за виконання відповідає AMD, компілятором від машинного коду до зрозумілого є результату є hip, а інструментами HIPIFY (перекладач з CUDA до AMD), ROCm CMake (проектний будівельник), ROCm Badnwidth Test (перевірка швидкості між кількома GPU), AMD SMI (візуальна приладова панель), бібліотеками hibCUB (сортувальник та організатор масивів даних), half (16-бітний зберігач пам'яті), RCCL (комунікація між GPU), Composable Kernel (набір інструментів, що спрощують розробку AI-операцій) [19] тощо.

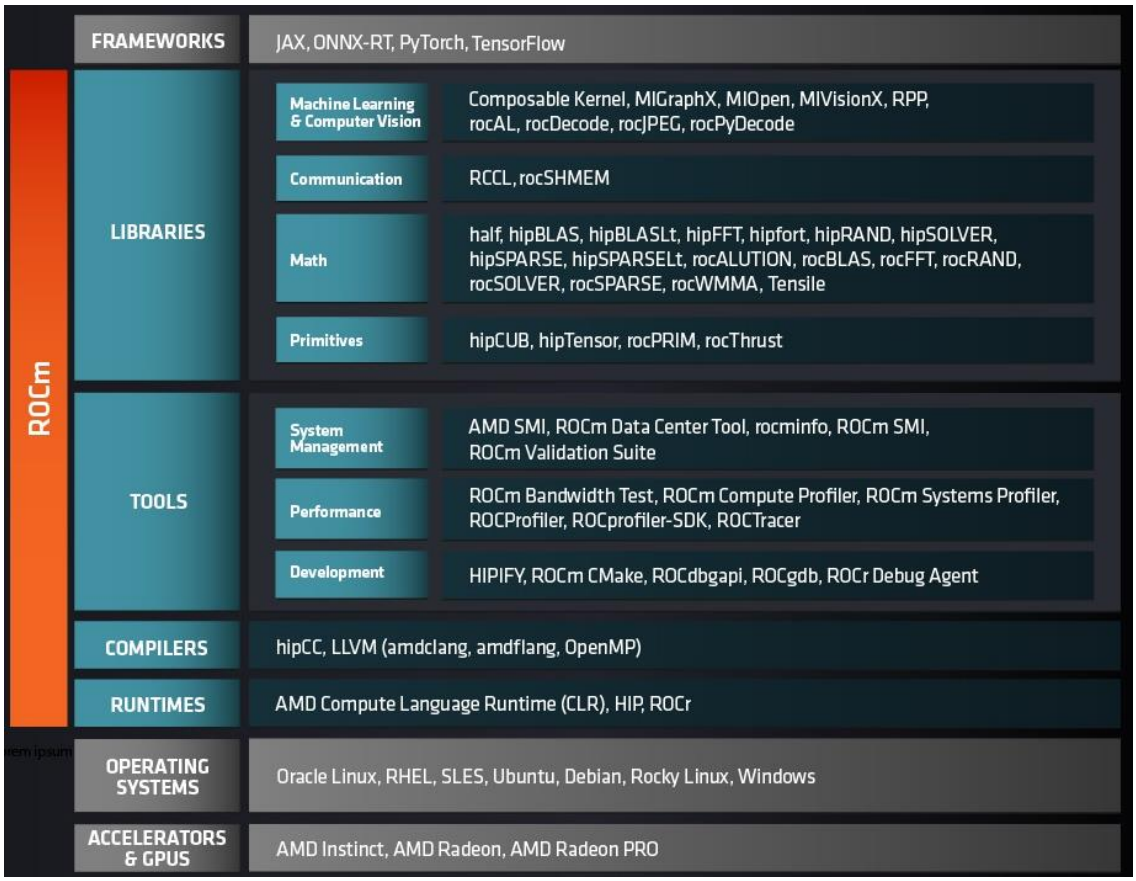


Рис. 3. Стек технологій платформи ROCm для апаратного прискорення алгоритмів машинного навчання

Розглянуті інструменти та бібліотеки допомагають спростити написання CUDA-коду (NVIDIA's C++ код, зрозумілий для GPU's). Проаналізований NVIDIA CUDA репозиторій стосується «проблеми індустрії фінансів, а саме

розбіжності між швидкістю обчислень та складністю моделі. Використовуючи прискорене обчислення NVIDIA, можна досягти в 160 разів більшої оптимізації, та в 100 разів швидшої сценарної генерації» [20]. Під прискоренням обчислень NVIDIA мається на увазі прискорення передзавантаження даних, а також вибірка розподілу результату.

Що стосується у 160 разів більшої оптимізації, використовується NVIDIA cuOpt (оптимізатор) open-source solver, який дозволяє ефективно вирішувати проблему обчислення середнього значення можливих збитків у портфоліо. Mean-CVar є покращенням Mean-Variance, що лише аналізує мінімально-допустимі збитки. Для тестування фінансової моделі використовується CUDA-X Data Science. Функцією цього інструменту є тестування торгівельних стратегій та покращує обрану.

В додатку А6, який є візуальною презентацією роботи описаного процесу, видно, як фінансова інформація потрапляє у базу даних CUDA-X Data Science, що повертає прогнози. На основі цих прогнозів генеруються сценарії та вираховуються цілі з обмеженнями. Mean-CVar вираховує середні можливі збитки з цих прогнозів – формується стратегія, яка надалі покращується завдяки CUDA-X Data Science HPC (High-Performance Computing) SDK (набір інструментів для створення користувацьких додатків машинного навчання на C++ та Fortran). Обчислений результат відправляється до фінансових менеджерів, які використовують цю стратегію, і на її основі вихідних даних цикл фінансової оптимізації починається спочатку.

ROCm дозволяє пришвидшити можливості відеокарти у 2-5 разів. Це досягається шляхом оптимізації вихідних гістограм обчислень LightGBM. Цей алгоритм градієнтного бустингу спрощує вибірку особливостей клієнтських ознак тим, що не потребує кодування «один до одного». Під типом цього кодування мається на увазі групування індивідуальних категорій за колонками, де істинність значення дорівнює 1, а протилежна відповідь – 0 (рис. 4). Кодування один одного особливо важливе для моделей, які не здатні працювати з назвами ознак, розрізняють лише числа.

Color	Red	Blue	Green
Red	1	0	0
Blue	0	1	0
Green	0	0	1

Рис. 4. Приклад попередньої обробки категоріальних даних методом One-Hot Encoding

За рис. 4, рядки в таблиці є ознаками, у той час як колонки – варіантами проаналізованої інформації.

1. 8. Роль SNN у валідації кредитної історії за пошуком схожості між користувацькими сценаріями

Як стверджує Х. Аміт (H. Amit) «Наприклад, у моделях випадкового лісу один до одного закодовані ознаки дозволяють утворювати з цих категорій окремі гілки. У алгоритмах, що маніпулюють відстанню, такими як SNN чи k-NN, кодування один до одного допомагає обчислити відстань між цільовими даними, без сортування категорій» [21]. За статтею В. Б. Кеннеді (W. B. Kennedy), алгоритм SNN є двоетапним процесом, який починається з обчислення необроблених попарних відстаней.

Після цього створюється друга матриця, з відстанями до найближчих сусідів. Пояснити парні відстані, слід звернутись до простого прикладу: нехай існує таблиця, у як є 100 елементів. Попарною відстанню є відстань між двома елементами, наприклад між 1 і 2, 1 і 97 або 40 і 67. Попарні відстані розглядають всі та об’єднуються між всіма елементами однаково, що призводить до формування таблиці парних відстаней у 10000 відстаней між 100-ма елементами. Під час першого етапу відбувається спарсифікація – на кожну пару відстаней a і b, створюється зв’язок з c-ти найближчих сусідів. Серед цих c сусідів розшукується a для b чи навпаки.

Внаслідок чого отримується матрицю з відстанню та кількістю сусідів, серед яких імовірно є протилежна парна відстань. Така колекція створюється як для a, так і для b. І в залежності від перетину цих двох колекцій, можна зрозуміти, наскільки колекція a перекриває колекцію b чи навпаки. Колекції можуть бути ще більш схожими, якщо вектори в їх колекціях мають не лише схожі відстані, але й схожий порядок. Наприклад, якщо для обох колекцій сусід А найбільший, потім В тощо. Для кластеризації SNN спочатку обчислюються відстані, потім використовується алгоритм DBSCAN, ідентифікуючи головні точки, знаходячи інші точки достатньо близько, щоб бути у одному ж кластері, і згодом вони поєднуються разом у кластери.

1. 9. Векторна бібліотека FAISS та її роль у системі аналізу кредитних ризиків

На нашу думку, інша назва таких алгоритмів – «векторні», оскільки векторні бази даних також працюють з дистанцією, «тяжінням» між інформацією. Прикладом такої бази даних, яка працює з фінансовою інформацією, є FAISS. FAISS є векторною бібліотекою з відкритим кодом, заснованою Meta, сфокусованою на вирішенні проблем кластеризації та пошуку за подібними елементами [22]. Як стверджує А. Меротра (A. Mehrotra) у своєму репозиторії на GitHUB, векторна бібліотека FAISS є ефективною у створенні фінансового агента, який обчислює та аналізує наповнення фінансових документів. Агент фільтрує, перевіряє інформацію зі збереженням контексту, та виводить результат, з подальшим прийманням запитів від користувача х [23].

Структурована база даних є неефективною у процесі такого пошуку, оскільки, на відміну від векторної, структурована складається з чітких зв’язків, що не тяжіють у довільну сукупність. Пошук відбувається за допомогою порівняння специфіки сукупностей векторів, за схожістю між ними. Ще однією векторною бібліотекою, як каже Д. Барклоу (D. Barcklow) є NMSLIB. Вона створена для того, щоб визначати відстані між векторами, навіть якщо вони неметричні, на відміну від FAISS, яка швидка, але покладається на метрики

симетричності та точність визначень відстані. Несиметричністю є ситуація, коли початкова точка не дорівнює нулю, а A -> B -> C може бути еквівалентна відстані A -> C. У статті автор стверджує, що NMSLIB працює з неметричними відстанями швидше. Бібліотека пропонує алгоритм HNSW (Hierarchical Navigable Small-World), у якому вершини об'єднуються векторами з найближчими сусідами, щоб сформувати графік векторів. Після цього, графік аналізується, щоб знайти приблизно найближчих сусідів відносно цільового вектора. HNSW завжди проходить через найближчого кандидата та його близьких сусідів [25]. У рис. 5, червоне коло означає початковий вектор у алгоритмі.

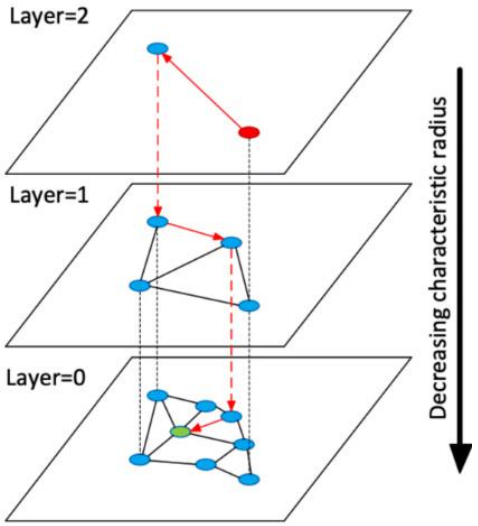


Рис. 5. Схема багаторівневої навігації та пошуку найближчих сусідів в індексі HNSW

Червоні стрілки показують напрямок жадібного алгоритму з початкового вектору до цільового. Цільовий вектор зображений, як зелене коло. Оскільки FAISS швидка, вона більше підходить до продуктивного середовища, а отже і майбутнього практичного використання.

1. 10. Кращі проектні практики: модульність, масштабованість, безпека

Векторна бібліотека FAISS разом з моделлю градієнтного бустингу LightGBM та інтерпретованістю SHAP є «ядром» проекту. На основі цього ядра додається другий, серверний рівень, у якому запити користувача поділені не ендпоїнти, зони відповідіальності для обробки вхідної інформації. Третій, клієнтський рівень, є графічним інтерфейсом, що забезпечує інтерактив з користувачем.

Розглянемо модульність такої архітектури. На нашу думку, модульністю є розділення коду на окремі файли (або одного об’єкту посиланнями на декілька), та експорт функціональності між ними. Мета такого підходу – забезпечення прозорості та зрозумілості коду, відокремлення бізнес-логіки. У цій статті [23] автор вважає, що систему аналізу кредитних ризиків слід оцінювати не за здатністю розділяти платників з-поміж боржників, а за якістю аналізу кредитної історії.

Історія не завжди є чесною, і деякі транзакції можуть виглядати аномально на фоні відсутності інформації за інші періоди. Для того, щоб пройти перевірку, боржник здатний на фальсифікацію власних даних, зміну банків для оновлення лімітів та короткочасну значну оптимізацію властивостей своєї кредитної історії. За статтею, базові кредитні моделі не здатні розрізнити дві ідеї, а виконують їх разом: визначають ризики, і також приймають рішення рішення на основі розрахунків цих ризиків. Щоб усунути цю проблему, система спеціально розділена на два шари:

1. Модельне обчислення ризиків
2. Механізм формування політик на основі правил

Обидва пункти мають власну функцію: перший пункт оптимізований для статистичної точності визначення ризиків; другий сфокусований на прийняття рішень, зрозумілий достатньо, щоб пройти кредитний аудит. Системи кредитного аналізу за такої ситуації побудовані на кредитних сигналах. Ними можуть бути

кількість і довжина кредитної історії, наявність злочинів, рівень освіти, досвід роботи, відношення боргу до зарплатні, та призначення кредиту. Враховуючи ці критерії, кредитна оцінка вже не є визначальним фактором. Отже, справжнім ядром проекту в цьому контексті виступає модельне обчислення ризиків. Без цього обчислення формування політик втрачає сенс, оскільки немає визначення клієнта як боржника. Сервер покладається на бізнес-логіку всередині ендпоїнтів – без них користувач не отримує необхідний вердикт, а без фронтенд-частини втрачає інтерактив та зручність експлуатації. Елементи архітектурного дизайну для ефективної масштабованості мають бути з початку дотримуватись кращих практик, оскільки лише зрозумілий код здатний до розширення. Разом з покращенням модульності, зменшується ризик зламу – «прозорий» код не приховує частини, де доступ до додатку є найбільш відкритий. Модульність архітектури спрощує підтримання безпеки та полегшує масштабованість.

РОЗДІЛ 2. ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ ДЛЯ ПОБУДОВИ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ

2. 1. Використання GAN у генерації синтетичних клієнтських даних

Незважаючи на повноту розглянутих інструментів, все ж є сфери діяльності у аналізі кредитних ризиків, які потребують подальшого аналізу. Такою сферою є генеративно-суперницькі мережі, скорочено GAN (Generative Adversarial Networks), які допомагають доповнити справжню клієнтську інформацію синтетичною. У реальному світі часто необхідної інформації буває недостатньо для неупередженого вердикту. Частина інформації стирається навмисно боржником, ще частина відсутня через оновлення банківської інформації. Якщо немає достатньо клієнтів зі схожим сценарієм для порівняння, замала кількість контексту додасть зайвий відсоток галюцинацій, непередбачуваність відповіді.

Щоб цьому запобігти, архітектура GAN поділяється на генератор та дискримінатор. Генератор трансформує 100-вимірний Гаусовий шум у синтетичну інформацію за допомогою функції-активації Tanh, тоді як дискримінатор використовує рівні SoftMax, щоб аналізувати одночасно реальні та синтетичні дані. Критерій перевірки залишається незмінний – чи має користувач високий відсоток дефолту. Завдання генератору – створити синтетичний екземпляр клієнтських даних, який не буде відрізнятись від справжнього. Дискримінатор знаходить та фільтрує інформацію, в залежності від її якості. Через деяку кількість тренувань межа між справжнім екземпляром та синтетичним зникає.

Внаслідок цього формується Еквілібріум Неша (Nash Equilibrium), де синтетична частина даних не може бути проаналізована без участі справжньої [27]. За схемою у додатку А9, справжня та синтетична інформація змішується разом для оцінки дискримінатором на дефолтність. Гаусовий шум проходить через лінійний генератор, тоді як спражні дані через образну конверсію. Результати з обох сторін змішуються у дискримінаторі. Якщо

дискримінатор розрізняє походження даних, він запускає «гру» з нульовою сумою – синтетична інформація буде перероблюватись у лінійному генераторі, поки не стане невід'ємною від справжньої. Повнота наповнення збільшує точність моделі завдяки присутності прикладів. Втім, ці приклади (неважливо, справжні це, чи наближені до справжніх) можуть слугувати зайвим шумом, коли йдеться про істинність деяких сигналів.

2. 2. Каузальні моделі, подвійне машинне навчання та п'ятикратне перехресне тестування

Таким сигналом є бажання клієнтів знайти роботу після отримання бонусу або покращити свою кредитну історію після отримання несприятливих результатів. Це сприяє утворенню питання «що, якщо я покращу показники збільшенням заробітної плати та сплатою боргу?» замість «чому я отримав такий результат». Друге питання цілком задовольняється використанням SHAP у додатку, а ось перше найбільше розкривається, коли йдеться про виведення причинно-наслідкових зв'язків (Causal Inference). За статею К. Мендеса (C. Mendez), «машинне навчання є ефективне у передбаченні, але це не стосується формування причинного ефекту.

Цілісність інформації може бути порушена факторами, які можуть впливати на тестування та результат з причин, які не відносяться до кредитної історії чи фінансової сфери. Для того, щоб запобігти ризику зникнення контексту, подвійне машинне навчання (Double Machien Learning, DML) відфільтровує шум, спричинений факторами, залишаючи лише причину в межах контексту. DML використовує дві машинні моделі (перша передбачує процес перевірки, друга прогнозує результат), виділяє залишки в першій та в другій, та об'єднує їх, щоб ізолювати від шуму причинний ефект. Щоб запобігти перенавчанню, DML залучує п'ятикратне перехресне тестування. Дані розділені на п'ять частин, де одна з частин схована, і застосовується для тестування моделей, які тренувались за іншими чотирма частинами» [28].

Причинно-наслідкові зв’язки відтворювались на основі пенсільванських даних про працевлаштування, де є дві групи: контрольна та бонусна (додаток 10). На відміну від бонусної, контрольна група не отримувала компенсацію за отримання роботи. Графік показує, що відсоток працевлаштованих у бонусної групи вищий на 0.09. Обидві групи приблизно однакові, з концентрацією працевлаштування між тривалістю 3 і 3.5. На тривалості від 1 до 3, а також від 3.5 до 4 присутні фактори, які не є чітко закріплені причинним ефектом, тобто мали інші причини для прийняття рішення. Це показує, що DML причинно-наслідкових зв’язків чітко показує вплив умов на розвиток значень.

Як стверджує Ф. Сіфуна (F. Sifuna), «каузальним висновком є статистичний фреймворк, метою якого є ізоляція конкретної властивості від загального шуму випадкових факторів. Каузальні моделі спрямовані на досягнення впливу втручання, чим відповідають на два питання: який середній ефект зростаючих кредитних лімітів серед усіх боржників, і як цей ефект варіюється серед різних сегментів боржників». Ф. Сіфуна пропонує додати до каузальної моделі «дані, що відносяться до соціального капіталу.

Соціальним капіталом є соціальні зв’язки, засновані на довірі спільноти. У невеликих кредитних історіях такі зв’язки є важливими. Боржники з високим рівнем соціального капіталу більш схильні до виплат, оскільки притримуються норм спільноти, та мають прийнятну репутацію». Каузальні висновки разом з даними соціального капіталу покращують чесність клієнтів. На нашу думку, забезпечують чесність вони тому, що у соціальному середовищі боржники можуть приховувати чи фальсифікувати показники кредитної історії, і погіршена репутація внаслідок несплати спонукає до дій.

2. 3. CreditXAI як кредитна система колаборації агентів та субагентів

Ієрархічна генерація результату на основі кредитної історії, та за допомогою координованої команди вбудованих агентів та субагентів є наступним рівнем розвитку системи аналізу кредитних ризиків. На думку Ю. Ші (Yu. Shi), Ж. Янг (Zh. Yang), Ї. Ванг (Yi. Wang) та Л. Жу (L. Zhou), «хоча додавання

нефінансової інформації збагачує дані та розвиває часткову інтерпретованість, моделям все ще не вистачає механізму ієрархічного мислення, що обмежує їх аналітичні можливості. Мультиагентна система Credit XAI вирішує цю проблему завдяки симуляції командного вирішення питань в процесі професійного кредитного аналізу. Командою у цьому випадку є сім кредитних агентів-аналітиків, де кожний агент має свою спеціалізацію» [30]. Ця мультиагентна система організована в три рівні спеціалізованих агентів:

1. Аналітичний рівень: агент аналізу фінансового ризику (FRA), ризику в бізнесі (BRA), офіційного урядового ризику (GRA), та агент формування комплексних рейтингів (CRA).

2. Рівень прийняття рішень: шеф з аналітики, та агент з написання рейтингових звітів).

3. Оркестрування: агент з системного менеджменту (додаток А11)

На нашу думку, агент третього рівня є головний керівник, який делегує зобов'язання між іншими агентами.

Як продовжують далі студенти Пекінського університету, «ще є четвертий рівень – Рівень обробки інформації. Цей рівень трансформує загальні дані у структуровані, які слугують вхідною інформацією для чотирьох агентів Рівня аналітики». Кожний з цих агентів генерує профіль клієнта з спеціалізованої точки зору (рейтинг кредитоспроможності клієнта та рівень ризику, що варіюється від 0 до 1).

Рівень прийняття рішень отримує ці чотири профілі та об'єднує їх у єдиний профіль, який відображає повну кредитну історію клієнта. Рівень оркестрування контролює весь процес від першого до третього рівня (Оркестрування є четвертим рівнем з абсолютним розтушуванням). Під «абсолютним розташуванням» мається на увазі відсутність прикріплення до конкретного рівня. Втім, CreditXAI має обмеження сфокусована саме на профілях клієнтів, хоча потенційно може залучити репутацію у соціальному середовищі, етичності, практиках менеджменту та соціальних зв'язках.

Загалом CreditXAI є чітким прикладом впровадження агентів та субагентів у кредитну систему. Ми вважаємо, що субагентами у CreditXAI є «агенти» другого та третього рівня, коли «агентом» четвертого рівня є справжній агент, який делегує завдання між субагентами, забезпечуючи бажану ефективність.

2. 4. Контрастне навчання на основі комбінованих ознак

Отримана інформація, яка отримується агентами, може бути оброблена на основі не лише індивідуальних ознак, але й комбінованих. Під комбінованими ознаками маються на увазі ознаки, які перехрещуються, та внаслідок цього утворюють нову ознаку. Комбіновані ознаки не лише ефективно репрезентують структуровані дані, а також більше збагачують семантичною інформацією, ніж індивідуальні. На нашу думку, під кращим збагаченням мається на увазі пропозиція з кращим контекстом, оскільки семантична інформація є головною суттю набору даних, який чутливий до загального контексту.

Перевагою контрастного навчання є можливість відрізнити семантичну інформацію від тієї, якій бракує контекст. У статті Ш. Оуянг (Sh. Ouyang), К. Бай (Q. Bai), Х. Фенг (H. Feng), Б. Ху (B. Hu) пропонують ідею «контрастного навчання для відстеження груп, що відмивають криптовалюту. Контрастне навчання знижує ефект шуму, що притягує дані того ж класу разом, у той же час відштовхуючи дані з інших класів. Алгоритм контрастного навчання базується на гетерогенній інформації з мережі Bitcoin, інакше кажучи, Bit-CHetG. Суть цього контрасту полягає у збільшенні схожості між векторами особливостей та нелегальними транзакціями, одночасно знижуючи схожість між особливостями легальних транзакцій» [31]. Двигуном цього алгоритму є формула контрастних втрат (2),

$$L_{CL} = -\sum_{m=1}^{M} \log\left\{ \frac{1}{|P(m)|} \times \sum_{p \in P(m)} \frac{\exp\left(sim\left(g_m, g_p\right)\tau\right)}{\sum_{n \in N(m)} \exp\left(g_m, g_p\right)\tau} \right\} \tag{2}$$

де вона є сумою усіх якорів m. На кожний якір береться один приклад зі списку позитивних, обчислюється, наскільки якір m близький до позитивного прикладу,

а також до всіх прикладів (негативних і позитивних). Обчислюється середнє з усіх позитивних прикладів, які схожі з якорем m (клієнтом). Опрацьовується негативний алгоритм, де вихідне значення низьке, якщо середнє значення високе, і навпаки. У додатку А13 чітко показана відмінність – негативні відношення мають збільшувати простір, в той час як позитивні – зменшувати, поєднуватись один з одним, формуючи семантичну інформацію, відокремлюючись від стороннього шуму. Проте, якщо відокремлена кількість властивостей сконцентрована не результаті, треба зрозуміти з якою точністю кредитний результат був наданий.

2. 5. Байєсівське машинне навчання та комбінований кредитний інтелект

Байєсівське машинне навчання здатне відповісти на питання точності наданого прогнозу. С. Наяк (S. Nayak) пропонує «Калібрований кредитний інтелект – систему, що комбінує Байєсівський аналізатор ризику для знайдення епістемічної невизначеності. Другим елементом у системі є модель градієнтного бустингу для контролю групових диспропорцій, зберігаючи продуктивність. І третім є стратегія злиття з урахуванням зсуву». Головною суттю третього елементу є стабілізація порогів прийняття рішень, що отримані від першого елементу [32]. Згідно алгоритму (рис. 6) початкові кредитні дані оброблюються перед потраплянням до байєсівського аналізатору, після чого проводиться перевірка на зсув та аудит, злиття, калібрація та пояснення. Стосовно байєсівського кроку 1, ініціюємо W (необхідні ваги нейроннної мережі), q_λ(W) як гаусівський розподіл як відсоток впевненості у результаті, λ як параметри розподілу, а λ від нуля – початкові значення.

Далі перераховуємо епохи, і з кожної епохи отримуємо набір релевантних даних. Пропускаємо параметри розподілу через ELBO функцію градієнтного спуск (мінімізація негативу) для того, щоб зробити поточне припущення W максимально наближеним до істинної оцінки. У другому кроці будуємо групу дерев прийняття рішень. Для цього створюємо порожній список дерев, і додаємо

до нього по дереву за кожний елемент з загального числа дерев. Поточний масив + додаткові дерева з кожною ітерацією проходять через функцію FitTree для тренування та запобігання перенавчання.

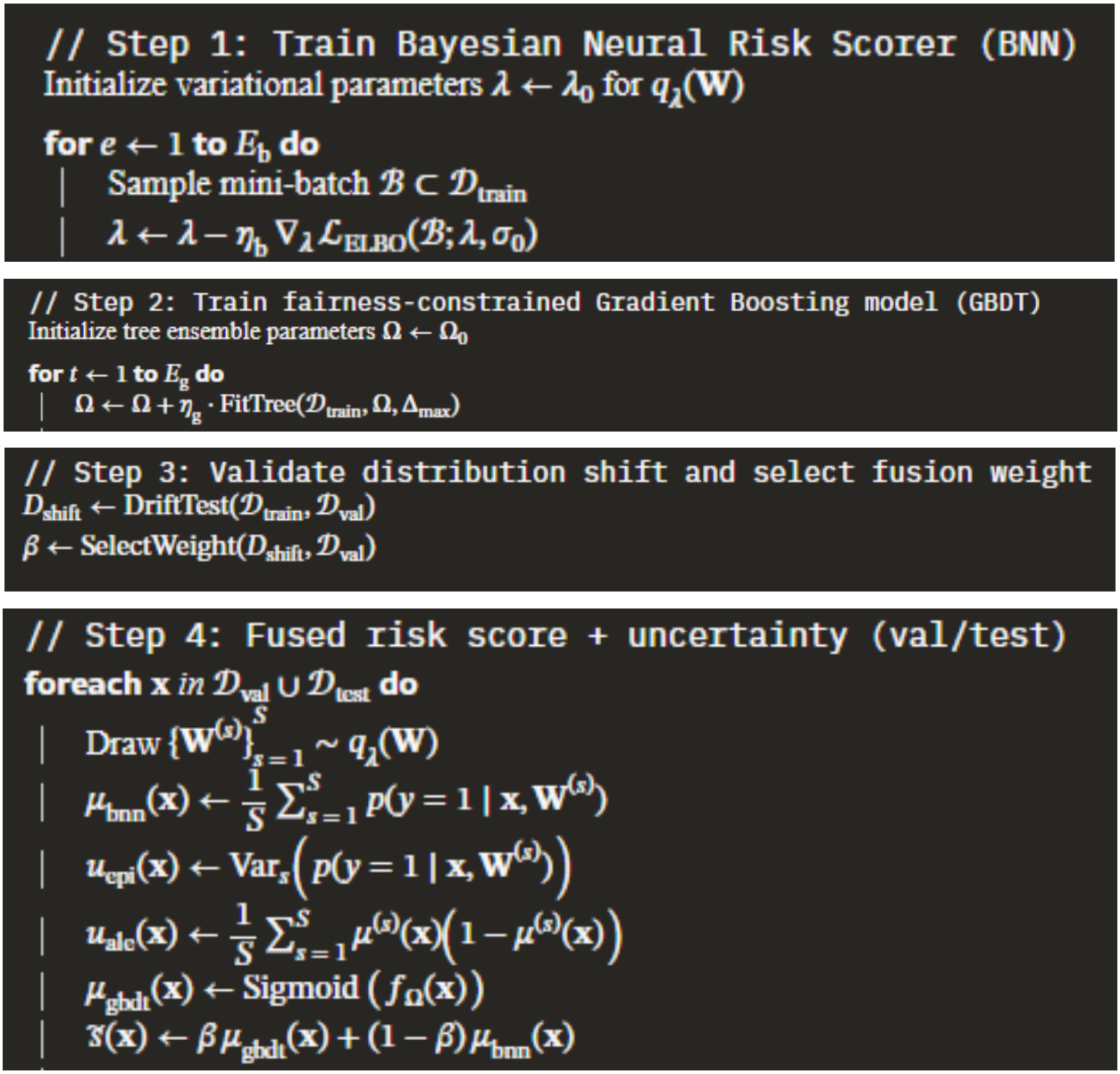


Рис. 6. Алгоритм розрахунку інтегрованого скорингового балу на основі GBDT та байесівської нейронної мережі (BNN)

Цей крок необхідний для чесності моделі. Суттю третього кроку є змішування Байєсівського аналізу з чесністю дерев прийняття рішень для того, щоб кредитні висновки залишались точними навіть при розбіжностях з реальними даними. Ми порівнюємо D_train (старі кредитні дані) та D_val (нові) та обчислюємо різницю між ними. Оптимізуємо це значення на основі функції

SelectWeight та ефективність моделей Байеса та дерев чесності. Це дозволяє Байєсівській мережі реагувати на зсуви економіки, в той час як градієнтний бустинг забезпечує чесність обчислення.

На четвертому кроці за кожней екземпляр даних у звичайних та тестових наборах ми комбінуємо Байєса та градієнт, а також визначаємо відсоток впевненості в результаті. З варіаційного розподілу створюється n нейронних мереж, кожна з яких знаходить можливість дефолту, а результат усереднюється. Далі обчислюється епістемічна неточність, де Байєсівська мережа ще не зустрічала схожий клієнтський екземпляр даних. Епістемічна неточність потрібна для того, щоб покращити ефективність люської присутності там, де моделі не можуть впоратись. Це також оптимізує архітектуру HITL (Human in the Loop).

Після цього обчислюється алеаторична неточність (неповні клієнтські дані) за допомогою можливості однієї модельної ваги, яка множиться на варіантність одного випробування Бернуллі (можливість дефолту). Далі результат обчислення градієнтного бустингу передається в сигмоїд, який повертає значення між 0 та 1 як можливість дефолту. Результатом є формула, яка множить вагу злиття β разом з градієнтом та (1-β) з байєсом. Коли значення β велике, перевага надається градієнту. Якщо навпаки, то байєсу. Отже, Комбінований кредитний інтелект допомагає визначити ймовірність дефолту через градієнтний бустинг, та епістимічну з алеаторичною неточністю.

2. 6. LendNova, мовна модель FinBERT та темпоральні вектори

Точність прогнозу може бути втрачена через відсутність повноти кредитних даних. Щоб запобігти цьому, існує технологія LendNova. Як стверджує автор, LendNova є End-to-End пайплайном для оцінки кредитного ризику, який працює безпосередньо за допомогою даних з кредитного бюро з використанням мовних моделей. LendNova минає ручну розробку функцій, яка характеризує традиційні підходи до кредитного скорингу. Система складається з трьох компонентів: етапу підготовки даних, конвертації в числа та прогнозування

задачі. Під час підготовки, LendNova конвертує дані кредитного бюро у структурні кредитні історії. Мовна модель (автор пропонує фінансову модель FinBERT), у свою чергу, перетворює ці кредитні історії у список чисел. Мовна модель надає перевагу часу – події за колишній місяць важливіші, ніж події, що сталися рік потому. LendNova планує знизити кошти та збільшити масштабованість на основі автоматизації попередньої обробки та отримання особливостей кредитної історії клієнта. LendNova конвертує економічні жаргонізми у окремі сегменти, які поділені за схожими кредитними особливостями. Сирі економічні дані мовна модель FinBERT перетворює у «кредитні історії» на розмовній англійській [33]. Модель використовує темпоральні вектори, щоб відокремити релевантні дані за часом – новіші ближчі, в той час як застарілі дані тримають дистанцію.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ АНАЛІЗУ КРЕДИТНИХ РИЗИКІВ

3. 1. Визначення та побудова архітектури

Згідно з практичною частиною роботи, робота над інтелектуальною системою кредитних ризиків починається з її архітектури. Початковою ідеєю було створення клієнт-серверної архітектури з ядром машинного навчання всередині серверу. Втім, під час розробки архітектура не підтримувала масштабованість, збільшувався ризик створення «об’єкту Бога», де увесь функціонал сконцентрований у одній сутності.

Внаслідок цього було вирішено створити мікросервісну архітектуру, де кожна особливість має власний сервіс. Стосовно клієнту, він залишався незмінний. Інтелектуальна система аналізу кредитних ризиків допомагає полегшити вирішення питання оцінки потенційних боржників, аналізуючи питання за допомогою сервісів, і всі ці сервіси об’єднані через єдиний шлюз API, який, у свою чергу перенаправляє запити з клієнта у конкретний сервіс. Шлюз має перевагу безперебійності – якщо один сервіс перестане працювати, це не стосуватиметься інших. Шлюз також слугує як проксі між клієнтом та кінцевими сервісами, знижуючи можливість знаходження та машинної моделі.

Архітектурну схему можна розділити за спеціалізацією: сервіси, які отримують дані за допомогою машинного навчання (власне сервіс кредитних аналізу ризиків за допомогою градієнтного бустингу LightGBM, сервіс пошуку подібної кредитної історії серед боржників використовуючи векторну бібліотеку FAISS, а також сервіс, який генерує синтетичні дані, схожі з клієнтськими, для кращої відтворюваності, за допомогою генеративних мереж CTGAN), сервіси прийняття рішень, уособлення бізнес-логіки та організаційних правил (політики прийняття рішень, організація процесів за допомогою оркестрування, збагачення даних, генерація звітів для документації), інфраструктура (шлюз API, Redis для короткочасного зберігання даних, Kafka для асинхронного зв’язку між сервісами). Клієнт же складається з лендінгу, метрик та документації. Перша та

третя сторінки слугують ознайомленням з проектом, у той час як метрики виконують головну функцію. Локальна збірка проекту стартує за адресою 3000, у той час як шлюз на 4000.

Мікросервіси, втім, мають власні адреси – від 8001 до 8009: аналіз кредитного ризику 8001, знаходження боржників 8002, сервіс прийняття рішень та керування 8003 та 8005, збагачення даних 8006, генерація синтетичних даних 8007, доповнений кредитний аналіз 8008 та аналіз соціального капіталу (репутації) на 8009. Клієнт побудований на основі бібліотек React 19 та Next.js 16 разом з App Router. Стилізація виконана через Tailwind CSS v4, а завантаження вхідних даних через TanStack Query. Серверна частину виконана за допомогою фреймворку FastApi на основі мови Python.

Для градієнтного бустингу моделей машинного навчання використовується LightGBM, FAISS для пошуку за схожими векторами та Redis дла кешування. Інфраструктурою побудованої кредитної системи слугує docker-compose збірка. Головним двигуном системи є модель, навчена на основі даних «Give me Some Credit». Альтернативою було створення на основі «German Credit Data». Приклад роботи, «happy path»: користувач заповнює форму, натискає «Predict (спрогнозувати)» - Tanstack Query отримує дані та передає їх до адреси шлюзу :4000/client-predict. Шлюз перенаправляє отриману інформацію вже до серверної кінцевої точки, до :8001/client-predict. На основі вхідних даних генерується відповідь, яка повертається назад через шлюз до користувача у вигляді графіків з числовими SHAP-результатами.

Повний цикл розробки, конфігураційні файли мікросервісів та клієнтського додатку розгорнуто в межах єдиного репозиторію. Ознайомитися з повною програмною реалізацією інтелектуальної системи аналізу кредитних ризиків можна у відкритому репозиторії Github: https://github.com/Hakkarian/CredsCore.

3. 2. Файлова структура клієнтської частини

Клієнт складається з папок app, components, hooks, lib, providers, public, tests та файлів налаштувань. Папка app є носієм сторінок клієнту, контейнера layout та налаштувань стилів. Layout завантажує необхідні шрифти, seo- мета-інформацію, огортає додаток у query провайдер для зв'язку з сервісами та забезпечує адаптивний дизайн згідно контрольних точок. Головна сторінка (лендінг) використовує динамічні імпорти для «лінивого завантаження» додатку. Складається зі статистик та особливостей додатку. У той час як головна та документаційна сторінки не мають основного підключення до серверу, сторінка з метриками концентрує сервісний функціонал.

Клієнтські компоненти згруповані за особливостями сервісів: agentic відповідає за аналіз кредитної історії за допомогою агента з субагентами, causal для причинно-наслідкових зв'язків кредитного результату, insights для комбінованих аналітичних висновків, landing для компонентів головних сторінок, landing для контейнеру та обгорток додатку, social для кінцевих точок серверу, пов'язаних з підтвердженням соціальної репутації. Synthetic-data відповідальна за компоненти, що є обгортками для згенерованої інформації на основі клієнтської. Папка ui є загальною та охоплює всі клієнтські компоненти, з метою перевикористання компонентів, що часто зустрічаються.

Серед компонентів метрик використовуються форма вводу, панель з результатами, панель з клієнтами зі схожою кредитною історією (відсоток схожості) та карусель з аналогічними результатами, дошка з моніторингом дрифту моделі, компонент зі схожими групами клієнтів. До метрик також входять компоненти з графіками ймовірностей, факторів ризику та статистик.

Папка lib включає в себе прикладні програмні інтерфейси, typescript-типи, та допоміжні інструменти. Щоб створити нову сторінку, достатньо додати її в директорію app/reports з іменем page.tsx. Додавання компоненту відповідає тим же принципам – створення у директорії components, імпорт компонентів де потрібно, та використання з псевдо- @/ чи відносних шляхів ../../. Ключовими

правилами є створення по одній сторінці (або панелі) на один маршрут в API шлюзі. Файли в nextjs за замовчуванням є серверними – для додавання клієнтського інтерактиву, до шапки файлу слід додати стрічку “use client”. Динамічні імпорти використовуються за прискорення навантаження при відкритті додатку.

3. 3. Next.js App Router та конфігурація маршрутів.

Next.js 16 використовує App Router для маршрутизації додатку: папки є адресними шляхами, в той час як page.txt – контентом всередині цих адресів. Next.js використовує next/font для автоматичної оптимізації шрифтів. Оновлення метаданих та QueryProvider у якості обгортки – все це відбувається всередині макету сторінок. Під час завантаження компоненту використовується React Suspense для появи іконки завантаження.

За адреси клієнтських сторінок відповідають файли page.tsx, а саме об’єкт, який компілює файли у маршрути. Наприклад, відносна адреса «/» є «app/page.tsx», а шлях «/dashboard» веде до «app/dashboard/page.tsx». Стосовно глобальних стилів, вони використовуються за допомогою tailwindcss для швидкої стилізації, а модульний препроцессор scss дозволяє ізолювати стилі у окремих компонентах/сторінках. У tailwindcss конфігураційному файлі застосовуються теми кольорів та параметри переходів анімації. Tailwindcss застосовує префікси в класах для розмежування логіки в класах, до прикладу bg- (фон), text- (текст), rounded- (радіус межі), а також скорочення понять flex та grid.

Модульність препроцесору scss дозволяє ізолювати стилі компоненту – стилі зберігаються у окремому .module.scss файлі, у вигляді об’єкта з властивостями, кожний з яких відповідає окремому класу. Щоб додати стилі до компоненту, слід лише додати до елемента атрибут className, значенням якого є ім’я об’єкта з класами, та необхідний клас в якості властивості цього об’єкта. Міксини також полегшили стилізацію шляхом перевикористання css-правила між іншими класами, наприклад: scss @mixin hover-lift { transition: all 0.3s ease;

&:hover { transform: translateY(-2px); box-shadow: 0 8px 25px rgba(0, 0, 0, 0.2);}} .card {@include hover-lift;}.

Препроцесор scss дозволяє застосовувати клас всередині класу, це пришвидшує розробку. Проект комбінує обидва інструменти – tailwind для компонентів, які застосовують мінімум стилів; модульний scss для більш ґрунтовної стилізації. Присутня палітра кольорів: кнопки мають золотий відтінок, фон сторінки темно-зелений, картки світло-брунатні. Кольори помилок, тривоги та успіху мають червоний, жовтий та зелений кольори відповідно.

Щодо відступів, вони зроблені за допомогою одиниці rem, що складає 16 пікселів у корені стилізації. Де xs – 0.25rem (4px), sm – 0.5rem (8px), md – 1rem (16px), lg – 1.5rem (24px), xl – 2rem (32px). Типографія шрифтів має приблизну метрику, зокрема text-xs (0.75rem), text-sm (0.875rem), text-base (1rem), text-lg (1.125rem), text-xl (1.25rem), text-2xl (1.5rem), text-3xl (1.875rem), text-4xl (2.25rem). Всі кольори задовольняють вимоги WCAG, для інтерактивних елементів присутні індикатори фокусу.

3. 4. Клієнт-сервісна комунікація за допомогою API

Як вже було зазначено вище, клієнтські запити, перед тим як опинитись всередині сервісів, мають пройти через шлюз API, що розташований за адресою :4000. Клієнт організований за сервісами, які відповідають за конкретний сервіс: class ApiClient { async get(path: string) {} } (приклад клієнту з HTTP логікою). А також класи-кінцеві точки кредитного аналізу, пошуку за подібностями чи комбінованого аналізу.

Замість того, щоб вручну керувати станами завантаження (React Suspence), на проекті використовуються функції-хуки Tanstack Query, як наприклад useHealthCheck: function useHealthCheck() { return useMutation({queryKey: [‘health’], queryFn: () => api.health.check(), refetchInterval})}Налаштування next.js адаптовані для контейнеризації docker, в той час як typescript використовує strict мод та псевдошлях для скорочення імпортів, наприклад «../../pages» стає

«@/pages». Команда «use client» використовується для клієнтських функцій-хуків, браузерних API, чи обробників подій, як наприклад onPress чи onClick.

Після створення цієї функції-хуку, її можна викликати всередині клієнта, у компоненті метрик: const health = useHealthCheck(); і на основі результату перевіряти, чи сервісні кінцеві точки працюють. Оскільки Tanstack Query повністю охоплює зв’язок клієнта з API шлюзом, query може витягувати інформацію, та на її основі рендерити відповідні частини компонентів або сторінки. Приклад повного контакту клієнта з API шлюзом: компонент викликає Query (нехай predict.mutate(data) – mutate означає надіслати запит з POST чи PUT методом); функція-хук викликає api.credit.predict(інформація від компонента, data); клієнт надсилає запит на адресу шлюзу, який передає сервісу з цією кінцевою точкою, нехай /client-predict.

Кредитний сервіс запускає модель градієнтного бустингу LightGBM; результат повертається назад до клієнтського компонента. Клієнтський API відловлює помилки, та замість них показує відповідні повідомлення. Результати автоматично зберігаються (кешуються) за допомогою Tanstack Query, при цьому в параметрах можна вказати час, скільки інформація залишатиметься релевантною, поки не зникне з кеша. Внаслідок цього інформація показується відразу, швидко оновлюється та не дублюється логікою всередині компонентів.

3. 5. Typescript і проектна типізація.

Що стосується типізації на проекті, вона влаштована за допомогою typescript інтерфейсів. Інтерфейси можна назвати кресленням об’єкта, класу чи компонента. Інтерфейс складається з типізації параметрів компонента. Наприклад, компонент приймає вік та платню клієнта за місяць. Без typescript компонент не знає, який тип має вік та платня – стрічка це, чи число. Інтерфейс допомагає затвердити тип, саме той який потрібно.

Завдяки такому захисту додаток не зламається внаслідок обчислення результатів разом з випадковими стрічками, undefined чи символами всередині. Окрім примітивів, інтерфейс приймає також масиви чи об’єкти інтерфейсів в

якості параметрів. Якщо інтерфейси мають ідентичну назву, при збірці проекту обидва інтерфейси зливаються воєдино, зберігаючи параметри всередині.

При типізації коду, синхронного чи асинхронного, дотримуються одного правила – як тип, any дозволяє використовувати будь-які типи, тобто any «вимикає» послуги typescripty. Any подекуди буває корисним у випадках коли йдеться про термінове та короткочасне вирішення проблеми, проте лише в якості unknown. Наприклад, результат обчислень as unknown as number. Такий спосіб змушує надати змінній більш точний тип, оскільки unknown є «невідомим типом», він не є типом, який охоплює всі інші. Typescript відловлює помилки, поки код працює.

В системі аналізу кредитних ризиків змінні мають точно відображати власні типи, щоб запобігти створенню майбутніх проблем з обчисленням. Разом з захистом типізації, Typescript покращує документацію проекту. Прикладом документованого сервісу можна вважати інтерфейс ApplicantData та його параметри: використання кредитного ліміту, вік, погашення кредиту у проміжку між 30-59 днів, відношення боргу до платні, місячна платня, загальна кількість акаунтів, кредит не сплачувався вже 90 днів, несплата за іпотеку, несплата кредиту у проміжку 60-89 днів, а також кількість утриманців.

У відповіді сервісу аналізу кредитних ризиків маємо інтерфейс PredictionResponse з параметрами prediction (чи ризиковий клієнт, по замовчуванню true), default_probability (ймовірність дефолту), rist_level (ступінь ризику [«Низький», «Середній», «Високий»]) та фактори ризику (масив інтерфейсів з особливістю клієнта, число позитивного чи негативного впливу, а також чому фактор ризиковий). Typescript допомагає вирішити помилку відсутності даних – іноді відповідь може дорівнювати або числовому значенню, або undefined.

У такому випадку, або після змінної типізації потрібно поставити знак питання (означає, що ми не впевнені чи прийде значення), або явно визначити, що приходить або число, або null (початкове значення). Історично склалось, що значення null є об’єктом. Всередині, Javascript-значення складаються з набору

бітів. Троє з цих бітів належать типу значення, решта є власне даними. Тип об’єкта є еквівалентним до трьох нульових бітів. Оскільки у Null C++ поінтер дорівнює нулю, значення null, відповідно, є об’єктом.

Мови Java та C++ мають схожу помилку з null. Оскільки кожний сайт всередині має перевірку на null (система аналізу кредитних ризиків не є виключенням), вирішення цієї проблеми спричинить появу більш критичної. Крім необов’язкових типів, typescript приймає типи union (отримання лише одного з перелічених значень). Що стосовно типізованих значень сервісів, сервіс пошуку за подібними клієнтами приймає ідентифікатор клієнта (стрічка), оцінка схожості (між 0 та 1), числова дистанція між векторами клієнтів, та власне особливості кредитної історії.

Як результат, присутні клієнти зі схожою історією, середня дистанція між ними (рівень схожості), та яка концентрація векторів у одній точці. Якщо клієнт перевірений за даними боржників та має схожу історію, результат висвітлюється інший, а саме булеве значення підозрілості, числова схожість, стрічковий масив з кредитними історіями інших клієнтів, а також зловмисницьке кільце, можливість знайдення якого за умовою підозрілості історії. Якщо кільце знайдене, і також всі хто до нього входять.

Сервіс прийняття рішень приймає типізований запит – результат прогнозу (інтерфейс кредитного аналізу), числова кількість позики, термін дії, а також зацікавленість в ній. Стосовно результату політик, типізованим є кортеж decision, який приймає Approve (дозвіл), Review (перегляд), Decline (відмова). Також ступінь оцінювання, теж кортеж, що приймає A | B | C | D | E як конкретний бал. Відповідь з політики також включає в себе дозвіл на фіксовану позику, умова отримання, а також пояснення прийняття такого рішення.

Типізація каузального сервісу, як й інші, покладається на опрацювання запиту, та отримання відповіді. Відповідь є масивом результатів у випадку альтернативного методу, і також ступінь впевненості у цьому майбутньому. Запит складається з числової ймовірності лікування кредитної історії, масиву інтерфейсів-ефектів (стрічкове лікування, числова сила цього лікування, відстань

між двома точками – інтервал впевненості, а також метод – порівняння, ваги, чи комбінація цих двох методів).

Інтерфейс Counterfactual Outcome відповідає на основне питання каузального сервісу – що було б, якби в кредитні історії специфічні особливості були розташовані *інакше.* Це альтернативний вимір, всередині якого клієнт розуміє як почати покращення власної кредитної історії. Типізований інтерфейс Counterfactual Outcome приймає стрічковий сценарій (більш висока платня, оптимізоване співвідношення позика/платня відповідно), видозмінені особливості кредитної історії з сервісу кредитного аналізу, числові нова ймовірність та зміна ймовірності. Типізована відповідь аналізу соціального капіталу кредитної історії складається з ідентифікатору клієнта, інтервалу соціалізації між 0 та 1, метрик соціальних мереж, а також візуалізації співвідношення репутації клієнта в залежності з його позикою.

Якщо потрібний більш точний, розгорнутий звіт, Typescript допомагає поєднати кілька інтерфейсів у один, як наприклад поєднання відповіді з сервісів кредитного, каузального та соціального аналізу. На проекті також перевикористовуються типи-«дженерики» – вони потрібні для створення шаблону, за яким створюються екземпляри специфічних типів. Прикладом цього є ApiResponse<T> (де T є параметром, що приймає довільний тип). Ми вважаємо, що «дженерик» є типом-функцією від Typescript, яка приймає параметр на вході, а на виході повертає тип, потрібний для конкретного кредитного сервісу. Tanstack Query підтримує Typescript по замовчуванню, що спрощує роботу з ним.

Завдяки Tanstack Query, результати кешуються та ревалідуються у реальному часі, а запити досягають шлюзу API успішно. Основною функцією шлюзу API є маршрутизація за шляхом, що забезпечує словник SERVICE_ENDPOINTS, який перераховує кінцеві серверні точки адрес (наприклад /similar-applicants, /fraud-rings чи /client-predict)/). Схема роботи клієнт-сервісного зв’язку: запит відправляється за адресою /client-predict, проходить через шлюз до відповідного сервісу, який повертає відповідь. У цій

схемі є декілька переваг: сервісу можуть масштабувати або видозмінити без необхідності зміни в клієнтській логіці.

Шлюз вирішує проблему міждоменного обміну ресурсами (CORS), внаслідок чого зникає проблема дублювання логіки між сервісами. І також клієнту треба знати лише адресу шлюзу, а не численних сервісів. Шлюз також дозволяє регулювати балансувальників навантаження в залежності від здоров’я сервісів. Важливу організаційну роль, крім шлюзу API, має оркестратор – він комбінує, забезпечує комплексну роботу сервісів, комунікацію між ними.

РОЗДІЛ 4. РОЗГОРТАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

4. 1. Диригент мікросервісної архітектури.

Кредитна система розділена на 9 сервісів: прогноз результату залежно від кредитної історії; пошук підозрілої закономірності серед клієнтів; прийняття рішень, бізнес-логіка; координація дій; генерація допоміжної інформації, подібної до клієнтської; створення звітів; додавання додаткої інформації для уточнення результату; комбінація декількох оцінок; аналіз соціальної мережі; маршрутизація запитів. Що потрібно розуміти, клієнт має зв’язок лише зі шлюзом API.

Шлюз передає клієнтські запити до сервісів, які спілкуються між собою за допомогою оркестратора, зі спільною логікою, яку може використати кожний сервіс. Приклад координації з кількома сервісами: система кредитних ризиків збагачується додатковими даними, результат аналізу переходить до перевірки на підозрілість. В незалежності від результату перевірки, приймається рішення, та отримується клієнтом. Перед тим як продовжити, кожний крок чекає на виконання попереднього. Кожний сервіс має перевірку на здоров’я, чи спроможний сервіс доєднатись до клієнта відповідно до клієнтських запитів. Приклад відповіді кінцевої точки health у json-форматі: {“status”: “healthy”, “service”: “credit-scoring”, “timestamp”: “2026-01-15T10:30:30 (дата)”}.

Для всіх сервісів доступні типи для типізації, клієнт http та допоміжні інструменти для конфігурації сервісів. Кожний сервіс огорнутий в Docker-container, суть якого ми пояснимо пізніше. Сервіси можуть бути масштабовані автономно і горизонтально, крім оркестратора. Наприклад, запускається лише три екземпляри сервісу кредитного аналізу для підтримання високого трафіку, в той час як оркестратор залишається один та контролює процес. Зв’язок між сервісами асинхронний, за допомогою async/await. Під час розробки відбувається перевірка сервісних методів (connection polling) за циклом while та статусом «completed».

Якщо статус підтверджений, цикл розривається, якщо ж ні, перевірка чекає секунду, перш ніж почати заново, поки статус не буде «completed». Запити, які не завершились успішно, запускаються заново, з експоненційною затримкою (exponential backoff – 1, 2, 4, 8 etc). На період розробки було створено три скрипти локального запуску проекту: запуск усіх сервісів (start-all.sh), вимкнення сервісних процесів (kill-ports) та перезапуск (restart-all), який комбінує запуск та вимкнення сервісів.

4. 2. Сервіс аналізу кредитних ризиків

Слід почати з сервісу аналізу кредитних ризиків. У його основі зображений кредитний предиктор – клас, який використовує модель попередньо навчену градієнтним бустингом LightGBM, а також SHAP (Shappley Additive ExPlanations)-пояснювачем на старті. Кредитна модель натренована на історичних даних клієнтських кредитних історій. У якості вхідних даних, модель приймає 10 особливостей з клієнтської кредитної історії, в той час як вихідними даними є оцінка можливості дефолту в залежності від якості вхідної інформації. Модель, shap-пояснення та порядок особливостей завантажуються у початку роботи.

```
class CreditPredictor:
    def __init__(self, model_dir):
        self.model = joblib.load(f"{model_dir}/lightgbm_credit_model.pkl")
        self.explainer = joblib.load(f"{model_dir}/shap_explainer.pkl")
        self.feature_names = joblib.load(f"{model_dir}/feature_names.pkl")
```

Рис. 7. Програмна реалізація класу `CreditPredictor` для завантаження моделей та засобів інтерпретації

Приклад обробки клієнтської кредитної історії: вхідні дані за допомогою Pydantic перевіряються на тип. Отримується список особливостей за порядком

пріоритетності. Градієнтний бустинг LightGBM приймає їх та повертає середню ймовірність кожної з особливостей вплинути на значення дефолту.

Ця ймовірність категоризується відповідно до трьох груп – низької (менше 0.3), середньої (0.3-0.6) та високої (більше ніж 0.6). В залежності від категорії генерується shap-пояснення щодо тих особливостей, які найбільше вплинули на розвиток дефолту. Якщо ризик низький, позика дозволяється автоматично. Середній бал викликає перевірку, в той час як високий спричинює відмову. Приклад shap-відповіді: об’єкт з властивостями прогнозу, ймовірності дефолту, рівня ризику та факторів ризику (які особливості вплинули найбільше, з рівнем впливу та описом).

Збільшення ймовірності підвищує ризик, зменшення – знижує ризик відповідно. Інтеграція Shappley Additive ExPlanations дозволяє покращити пристосованість до регуляторних вимог. Наступною кінцевою точкою у сервісі кредитного аналізу є /similar-applicants, яка використовує FAISS для обчислення приблизних сусідів цільового вектора, знаходячи історичні екземпляри клієнтів зі схожими особливостями. Це допомагає проаналізувати клієнтів, у який немає достатньо інформації для повноцінного прогнозу. Пошук боржників працює інакше, ніж прогнозування. Замість навчання за даними зловмисників, FAISS навчається за пошуком подібних векторів, де кожна кредитна історія перевіряється за подібною поведінкою боржника. /explain-denial задовольняє регуляторні вимоги завдяки генерації пояснень для тих клієнтів, яким доступ до позики відмовлено. /monitor-drift перевіряє різницю між поточними та вхідними даними.

Це слугує сигналом для того, що потрібно поновити тренування для моделі. За структурою, система кредитного аналізу складається з директорії /app (основна логіка машинного навчання) та /saved-models (моделі для прогнозу). В директорії /app, всередині предиктору завантажується lightgbm та shap моделі. Функція predict приймає клас предиктору, та вхідні клієнтські дані. Після прогнозування, категоризації та генерації пояснень повертається об’єкт з результатами, який вже переходить до main.py (логіка з маршрутизацією за

допомогою фреймворку FastAPI) по шлюзу API назад до клієнта. Згідно з роботи сервісу кредитного аналізу, для локального використання прогнозів моделі, а самі прогнози (лише сервер) займають 20-30 мілісекунд. потрібно два гігабайти пам'яті.

Крім директорій з основною логікою сервісу також присутні файли з сервісною конфігурацією, та докером. Докер (Docker) є інструментом, який дозволяє огорнути проект та всі його залежності у контейнер. Всередині контейнерного середовища проект ізольований від зовнішніх впливів, та може запускатись при будь-якому апаратному забезпеченні. Докер-контейнер створюється за допомогою образу (креслень контейнеру). У той час як генерація образу викликається за допомогою файлу Dockerfile, зі списком команд:

```
# отримання образу з контейнерного репозиторію на Dockerhub
FROM python:3.11-slim
# завантаження залежностей для градієнтного бустингу LightGBM
RUN apt-get update && apt-get install -y libgomp1
# робочий шлях для проекту, директорія всередині контейнеру
WORKDIR /app
# копія залежностей всередині директорії /app
COPY requirements.txt .
# завантаження залежностей
RUN pip install -r requirements.txt
# копія сервісу у цільову директорію
COPY shared/credscore_shared /app/shared
# завантаження залежностей сервісу
RUN pip install -e /app/shared
# копія внутрішньої логіки
COPY credit_scoring/app /app/app
COPY credit_scoring/saved_models /app/saved_models
# доступність контейнера до підключення за адресою :8000
```

```
EXPOSE 8000
# команда для запуску контейнера
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port",
"8000", "--workers", "4"]
```

Варто додати, що залежність libgomp1 дозволяє LightGBM паралельно оброблювати кілька запитів до сервісу кредитного аналізу, а модель завантажується ще на старті. Помилки оброблюються за трьома статусами – 503, якщо модель не завантажилась, 422 при неправильному наборі вхідних даних (помилка, яку відловлює Pydantic), та 500 помилка прогнозування.

4. 3. Розробка сервісу виявлення шахрайства

Сервіс виявлення шахрайства аналізує кредитні історії з метою знайти підозрілі закономірності шляхом пошуку подібності. У цьому сервісі модель тренується на за значеннями особливостей кредитної історії, а за діленням всієї історії на вектор серед групи векторів (подібних історій) в базі даних відомих шаблонів. Підозрілість вища, чим ближче вектор (поточна клієнтська історія) ближче до відомих шахрайських схем.

Щоб досягти цього результату, сервіс використовує векторну бібліотеку FAISS від Facebook AI для індексації мільйонів історій в базі, результати пошуків якісні, та знаходяться за мілісекунди. Distances, indices = index.search (vector, k=5), де k є кількістю найближчих сусідів. if min(distances) < threshold: return {“is_suspicious”: True} (якщо мінімальна дистанція між сусідами нижче заданого порогу (ознака підозрілої закономірності), така історія не проходить перевірку).

За структурою, Сервіс виявлення шахрайства має ідентичну структуру, як і сервіс аналізу кредитних ризиків: має директорії /app (маршрутизація в main.py та faiss_service.py) та /saved_models з індексацією векторів та їх метаданими. Всередині main.py кінцева точка /similar отримує запит з методом “post”, заголовком json-типу контенту, а також тілом запиту від клієнту та шлюзу API. /similar повертає відповідь, в якій вказана підозрілість поточного вектору, рівень

подібності, вектори-сусіди на спільному рівні, а також явність закономірності їх розміщення (шахрайське кільце).

Кінцева точка, яка відповідає за шахрайські «кільця», є /fraud-rings. Ендпоїнт створює графік клієнтів за їх потенційними закономірностями (однакові адреси чи одночасні дії). Завдяки цьому допомагає проаналізувати організовану злочинну діяльність, – якщо таких боржників декілька, і всі вони розділяють спільні ознаки, таке угрупування є «злочинним кільцем», не лише співпадінням. Як і всі ендпоїнти з методом post (додавання інформації до бази), /fraud-rings має заголовку типу контенту json з тілом запиту, що має бути оброблене всередині сервісу, перед отриманням відповідної відповіді. Відповідь складається з булевого виявлення шахрайського кільця, масив причетних, та ступінь впевненості у виявленій закономірності.

Шахрайські кільця проявляються у однакових даних, які додаються з невеликими змінами (однакова адреса, але до номеру телефону додається поступово +1). Сервіс будує графік зі схожими векторами, знаходить кластери з поєднаними особливостями цих історій, та маркує кластери, які за одноманітністю перевищують заданий поріг. Поріг є динамічним, та змінюється ~%5 залежно від нещодавніх кредитних історій. FAISS не підтримує активне видалення індексів векторів. Коли індекси стають застаріли по відношенню до нових даних, вектори видаляються за допомогою перебудови індекса векторів.

Для цього експортуються нещодавні кредитні історії (вектори) vectors = load_recent_applications(days=30). Оскільки дані можуть приходити в сотнях, а то й в тисячах (значення платні, років, спроб затримання боргу), скалярна функція вирівнює значення так, щоб всі вони впливали на пошук рівномірно scaler.fit(vectors). Після вирівнювання нових векторів, FAISS-індексація на їх основі перебудовується index = faiss.IndexIVFFlat, index.add(vectors). Як тільки індексація перебудована, faiss зберігає поточну збірку до моменту, коли вона застаріє, і процедура почнеться заново. Потрібно зауважити, що сервіс виявлення шахрайства відмовляє в кредиті незалежно від якості історії, якщо характерні закономірності були виявлені.

У сервісі розширеного оцінювання оцінка шахрайства з меншим пріоритетом доповнює кредитний аналіз. Оскільки FAISS має чутливі дані, доступ закритий лише для авторизованих користувачів. Оскільки поточна стадія проекта не має авторизацію, доступ наразі є відкритий до всіх сервісів. Сервіс виявлення ризиків, як і сервіс кредитного аналізу, також має докер-контейнер з аналогічним викликом зображення python:3.11-slim, робочою директорією /app, копією конфігурації бібліотек всередині /app та встановлення їх в цій директорії. У внутрішню докерську директорію копіюються також файли faiss індексації, та відкривається порт, що є еквівалентним порту сервісу faiss.

4. 4. Сервіс причинно-наслідкових зв’язків

В той час як faiss-виявлення шахрайства чи lightgbm-кредитного ризику пропонують факт дефолту як наслідок високого споживання позики, сервіс причинно-наслідкових зв’язків або каузальний сервіс відповідають на питання «що-якщо». Що, якщо клієнт знизить співвідношення боргу до платні, наскільки зменшиться ймовірність дефолту? Складність в тому, що один клієнт не може існувати відразу у двох сценаріях одночасно. Знаходження подібних клієнтських кредитних історій, та заповнення на основі їх альтернативного результату є головним завданням цього сервісу.

Каузальний сервіс визначає три основні фактори – визначення ключових причин, внаслідок яких стан кредитної історії покращився (наприклад, не абстрактна фінансова дисципліна, а числові характеристики платні); позитивність висновку, у кожної клієнтської кредитної історії є вірогідність отримати лікування; і також стабільність історії, що означає підвищення кредиту та дотримання його погашення у визначених термінах.

Прикладом запиту до каузального сервісу є кінцева точка /estimate-propensity з тілом запиту:

```
curl -X POST http://localhost:8006/estimate-propensity \
  -d '{
    # особливості з шансом вплинути на лікування та на дефолт
    "covariates": {...},
```

```
    # тип лікування - допомога з усуненням боргу
    "treatment": "credit_counseling",
    # особливості кредитної кредитної історії клієнта
    "applicant_features": {...}
  }'
```

Відповіддю сервісу є відповідний об’єкт з властивостями:

```
{
  # ймовірність клієнта бути в категорії тих, хто отримав і
лікування, і дефолт
  "propensity_score": 0.35,
  # концентрація клієнтів з відсотком лікування та дефолту,
достатня кількість для конкретного висновку
  "overlap_assessment": "adequate",
  # вага особливостей кредитної історії
  "feature_importance": {...}
}
```

Для запуску кінцевої точки /estimate-treatment-effect потрібно надати тіло з властивостями treatment: higher_income, а також method: matching. Це означає, що лікування кредитної історії є за допомогою більш високої ставки, за подібними випадками лікування. В той час як метод IPW зважує аналізу ймовірності лікування, a doubly robust діє гібридно. У відповіді клієнт отримує:

```
{
  # середній ефект від такого лікування – зменшення дефолту на
5 відсотків
  "average_treatment_effect": -0.05,
  # впевненість у тому, що підвищення платні вилікує від 10 до
1 відсотка дефолту
  "confidence_interval": [-0.10, -0.01],
  # значення нижче 0.05 означає, що ефект існує
  "p_value": 0.02,
  # пошук за схожими випадками
```

```
  "method": "matching"
}
```

Кінцева точка /counterfactuals має наступне тіло запиту:

```
curl -X POST http://localhost:8006/counterfactuals \
  -d '{
    # повна клієнтська кредитна історія
    "applicant_data": {...},
    # альтернативний результат
    "interventions": [
      {"income_increase": 2000},
      {"debt_reduction": 0.3}
    ]
  }'
```

Відповідь, яка приходить з /counterfactuals:

```
{
  "counterfactuals": [
    {
      # сценарій, де місячна платня підвищена на 2000
      "scenario": "monthly_income_increased_by_2000",
      # поточний дефолт 72 відсотки
      "current_outcome": 0.72,
      # альтернативний (з покращенням платні) – 62 відсотки
      "treated_outcome": 0.62,
      # зниження дефолту на 10 відсотків
      "improvement": 0.10
    }
  ]
}
```

В кінцевій точці тіло /uplift за методом post (як й інші ендпоїнти), приймає sure things (альтернативний ефект гарантований), lost causes (немає ефекту), persuables (динамічна можливість зміни ефекту в залежності від лікування), sleeping dogs (лікування має шанс підвищити дефолт, не знизити).

Причинно-наслідкові сервіси впливають на кредитний аналіз (3):

0.6 * кредитний аналіз + 0.1 * каузальний сервіс + 0.3 * соціальний капітал(3)

Нижча вага кредитного аналізу означає, що перевагу має каузальний сервіс (більше пояснює «чому», ніж прогнозує результат). Каузальний сервіс може вирішити подібні сценарії: «що, якщо платня підвищиться на 1000, і дефолт зменшиться на 5 відсотків?», «перевірка на різноманіття лікування в залежності від групи», «якщо відмова в кредиті, що треба зробити для підвищення шансів уникнення дефолту?».

Втім, слід зауважити, що не всі особливості кредитної історії впливають на дефолт однаково. Наприклад, зниження не так впливає, як підвищення платні, чи уникнення штрафів за 90 днів пасивності, або збільшення кількості утриманців залежно від погашення боргу. Виконання кожної кінцевої точки по тривалості займає до секунди в залежності від інтенсивності обчислень, а також насиченості кредитної історії клієнта. Ідентичні запити зберігаються в кеші завдяки короткочасній базі даних Redis.

Ключовою особливістю каузального сервісу є відповідь на питання «що-якщо», менша властивість прогнозу, ніж пошук альтернативного, кращого результату. Оскільки цьому сервісу не є властивою точність, застосовується метрика впевненості у власній відповіді.

4. 5. Створення сервісу соціального капіталу

Стосовно сервісу соціального капіталу, його мета – зрозуміти клієнтську історію не як індивідуальну, ізольовану від громади, а скоріше об’єднану в соціальній мережі. В даному контексті соціальною мережею є набір клієнтських кредитних історій, пов’язаних разом соціальними зв’язками. Соціальний капітал

перевіряє, кого клієнт знає у власній мережі, хто з них кредитоспроможний, і як саме сусіди здатні забезпечити власну кредитоспроможність (вища платня, відсутність простроченості). Наприклад, переважна відсутність заборгованостей є у тих людей, які залучені у фінансові мережі, або також пов'язані з високою відповідальністю, організацією та делегуванням.

Даними для використання є як зв'язки між працівниками, так реферали. На соціальний капітал також впливають географічне розташування (клієнти, розташовані в одному районі чи в провінції, мають більше спільного, ніж люди за межами міста чи в межах однієї країни). Соціальний капітал також торкається систем сплати. Ця особливість є корінною, оскільки клієнт постійно проводить транзакції в залежності від поточного фінансового становища. Цифровізація фінансових систем дозволила сплачувати за певні послуги дистанційно. Це є причиною постійної релевантності соціального капіталу – клієнт з низькою репутацією схильний до великих транзакцій та має відповідний кредит.

Сервіс соціального капіталу знеособлює дані, внаслідок чого сервіс лише бачить пов'язаність фіксованого клієнта або групи з клієнтом номер два. Клієнти та зв'язки формують графік з вагами, вершинами та вузлами, де вагами є сила зв'язків (наскільки чіткими є). Вершини це власне зв'язки, а вузли є людьми чи групами людей, що уособлюють соціальну мережу. Сервіс соціального капіталу вимірює за допомогою формули number_of_connections (кількість зв'язків) / total_possible (загальна кількість), як багато соціальних зв'язків має клієнт, й чи має він слабкості в цих зв'язках.

Також стверджуємо, що людина займає важливу організаційну позицію, якщо об'єднує кілька груп, що не мають загальну тематику. Чи є зв'язки поєднаними з різних сторін? Наприклад, клієнт А поєднаний з клієнтом Б, який в свою чергу має зв'язки з клієнтами В і Г, й ті мають відгалуження у Д і Є тощо. Якщо така система присутня, тоді вона є надійною для проведення транзакцій, а отже й підвищення кредитних лімітів. Низька оцінка означає, що мережа є «розрідженою», з малим відсотком спорідненості. Така група може бути створена

тимчасово, формуючи шахрайське кільце, що є причиною відмови в незалежності від якості кредитної історії.

Щодо централізованості власних векторів, важливість соціального капіталу означає зв’язок з важливими людьми (кредитні історії, що мають довготривалий зв’язок з широким колом людей). Згідно тлумачень сервісу соціального капіталу, зв’язок з таким типом клієнтів є більш важливим, ніж поєднання з клієнтами, які не мають соціальних зв’язків, або мають ефемерні (короткотривалі з певною метою). Окрім зв’язку з коротривалими історіями, слід також зрозуміти, який відсоток з них має ризик дефолту, або взагалі отримали відмову в кредиті. Високе значення сусідського дефолту означає ризиковану для зв’язків спільноту, чи високу частоту появи шахрайських кілець. Оцінку 0-1 сервіс соціального капіталу комбінує за допомогою метрик. Прикладом цього можна вважати соціальну формулу social_score, яка поєднує всі особливості сервісу разом:

```
social_score = (
    # власні централзіовані вектори
    0.25 * eigenvector_centrality +
    # сусідська централізованість
    0.20 * betweenness_centrality +
    # вірогідність утворення спільноти
    0.20 * (1 - network_default_rate) +
    # ступінь централізованості
    0.15 * clustering_coefficient +
    0.10 * degree_centrality +
    # впливовість зв’язків
    0.10 * connection_strength
)
```

Чим вищий ступінь оцінки, тим вища позиція в соціальній мережі. У відповіді сервісу соціального капіталу присутні:

```
{
  # ідентифікатор клієнтської кредитної історії
  "applicant_id": "APL-123",
  # соціальна оцінка
  "social_score": 0.75,
  # метрики соціальних мереж
  "network_metrics": {
    # ступінь централізованості
    "degree_centrality": 0.32,
    # централізованість сусідів
    "betweenness_centrality": 0.15,
    # коефіцієнт кластеризації
    "clustering_coefficient": 0.68,
    # централізованість власних векторів
    "eigenvector_centrality": 0.45,
    # розмір соціальної мережі
    "network_size": 24,
    # середній рівень дефолту соціальної мережі становить 8%
    "network_default_rate": 0.08
  }
}
```

Щоб дізнатись детальну структуру соціальної мережі, відправляється запит на кінцеву точку /network-analysis, яка повертає відстеження спільноти по групам, індикатори ризику та відстеження аномалій для підозрілими закономірностями. Третій метод є спорідненим з сервісом виявлення шахрайства, оскільки обидва відстежуються патерни шахрайських утворень. Кінцева точка /visualization-data приймає ідентифікатор клієнтської кредитної історії, як запит,

та повертає вузли й вершини для бібліотек динамічної візуалізації даних для графік, наприклад D3.js. /risk-indicators повертає відповідь, яка є об’єктом з масивом indicators.

Масив складається із зауважень щодо підозрілих соціальних мереж. Кожне зауваження є об’єктом з властивостями type: suspicious_cluster (знайдене підозріле угрупування), severity: high (високий ризик), description: Tightly connected cluster with unusual patterns (угрупування тісне, централізоване, з аномальною поведінкою) або тип маленької соцмережі, з середнім ризиком та попередженням, що аналіз може бути неповний через недостатню кількість даних. Сервіс соціального капіталу показує, що тісні угрупування можуть бути синтетичними шахрайськими кільцями; аналізують незвичайно швидке формування зв’язків серед кредитних історій; висвітлюють різні топології, як наприклад топологія зірки (клієнт з розлогим відгалуженням сусідів).

Слід сказати, що результат цього процесу не є автоматичною відмовою в кредиті (за виявлення шахрайства відповідає відповідний сервіс), а лише поверхневою перевіркою соціальних мереж, що, враховуючи частку нижче, впливає на заключний результат. У покращеному кредитному аналізі одна третина припадає на сервіс соціального капіталу = 0.6 * (сервіси машинного навчання) + 0.1 * (каузальні сервіси) + 0.3 (соціального капіталу). Втім, клієнти з народження можуть перебувати у групі з незвичайною репутацією, штрафування за такі сценарії є ознакою упередженості сервісу.

Для протидії цьому аномальну соціальну мережу порівнюють з іншими нормалізованими групами; ізолюють сценарії, де має місце демографічна складова; дозвіл на пояснення власного становища; додавання пояснень на вплив соціального середовища на якість кредитної історії. Прийнятною практикою є неупереджена перевірка, розділена між сервісами, кожний з який аналізу одну з граней проблеми. Повний аналіз займає від кількох до десяти секунд: з них аналіз централізації сусідів має півсекунди (складність виконання - O (nm)), власні вектори 2 секунди – O(nm), пошук на упередженість до 5-ти секунд – O(nlogn).

Складність виконання O(n) означає, що механізм сервісу буде обчислювати результат невідому кількість кредитних історій.

При цьому, O(nm) означає, що буде перевірятись кількість клієнтських кредитних історій, та особливості, що до них належать. Наприклад кредитна історія A з платнею 5000. Під Onlogn мається на увазі, що перевірка ділить кількість клієнтів (чи кредитних особливостей) навпіл до тих пір, поки не буде знайдений результат. Сервіс соціального капіталу оптимізований для великих наборів даних (використовує вибірки), виконує обчислення паралельно, ідентичні запити закешовані.

4. 6. Сервіс доповненого кредитного аналізу

Незважаючи на обробку даних та повернення результату, каузальні, соціальні, та сервіси машинного навчання працюють лише в зоні власної компетенції, зрідка звертаючись у випадку шахрайських схем. Метою сервісу доповненого кредитного аналізу є об’єднання цих сервісів у єдину картину заради повернення результату, що повністю охоплює розглянуті вище аспекти.

Як вже було розглянуто, доповнений сервіс складається з трьох частин – сервіси машинного навчання (60%), сервіс причинно-наслідкових зв’язків (10%) та сервіс соціального капіталу (30%). Порівняно з іншими сервісами, машинне навчання кредитного аналізу має найвищий прогнозний потенціал у великих наборах даних; сервіс соціального капіталу доповнює машинне навчання додатковою інформацією про участь в угрупуваннях, а останні каузальні 10 відсотків пояснюють, чому клієнтській кредитній історії було відмовлено, і що потрібно зробити заради досягнення кращого альтернативного висновку. Висновок є емпіричним, тобто отриманим з набору даних, що застосовувався не для тренування, перевірений на основі фактів.

Отримання ймовірності доповненого кредитного аналізу визначається за допомогою формули:

```
augmented_probability = 1 - (
    0.6 × (1 - ml_default_probability) +
```

```
    0.1 × causal_score +
    0.3 × social_score
)
```

де фрагментом 1 – ml_default_probability є значення впевненості, що кредитного дефолту не буде, а глобальний 1 – (…) значення відсутності дефолту вже для доповненого кредитного ризику.

Це означає, що ризик є меншим з вищим числовим результатом. Сервіс доповненого кредитного аналізу визначає результат за допомогою класу:

```
class AugmentedScoringEngine:
    def __init__(self):
        self.credit_client =
# екземпляр сервісу кредитного аналізу
ServiceClient(os.getenv("CREDIT_SCORING_URL"))
        self.causal_client =
# сервіс причинно-наслідкових зв’язків
ServiceClient(os.getenv("CAUSAL_SERVICE_URL"))
        self.social_client =
# сервіс соціального капіталу
ServiceClient(os.getenv("SOCIAL_CAPITAL_URL"))

    # метод приймає сервіси, ідентифікатор та особливості
кредитної історії
    async def calculate_score(self, applicant_id, features):
        # сервіси викликаються паралельно
        credit_task = self.credit_client.post("/predict",
json={"client_data": features})
        causal_task = self.causal_client.post("/estimate",
json=features)
```

```
        social_task = self.social_client.post("/calculate",
json={"applicant_id": applicant_id})

        # Після того, як всі три сервіси отримали запит,
результат прийде не відразу – для цього використовується
синтаксис async await для використання асинхронного коду
всередині синхронного. Async використовується для визначення
типу функції, await за умови, якщо присутній async. Механізм
схожий з роботою Promise.all() Javascript – збір та повертання
результатів усіх присутніх асинхронних процесів.
        results = await asyncio.gather(credit_task,
causal_task, social_task)

        # з кожного результату (порядок елементів) отримується
оцінка
        ml_score = 1 - results[0].default_probability  # 0.75
        causal_score = results[1].potential_score       # 0.60
        social_score = results[2].social_score          # 0.70

        # і отримуємо знайому формулу з визначених оцінок
        combined = 0.6 * ml_score + 0.1 * causal_score + 0.3 *
social_score

        return {
            # можливість дефолту від доповненого сервісу
            "augmented_probability": 1 - combined,
            # вплив сервісів на оцінку
            "component_breakdown": {
                # машинне навчання
```

```
                "ml": ml_score,
                # причинно-наслідкові зв’язки
                "causal": causal_score,
                # соціальний капітал
                "social": social_score
            }
        }
```

Загальний час займає як один сервіс, тому що всі сервіси працюють паралельно, в кілька потоків. У випадку падіння сервісу, користувачу будуть представлені нейтральні оцінки сервісів:

```
if credit_service_down:
    ml_score = 0.5  # нейтральна оцінка
declining_factor += 0.6  # можливе відхилення впевненості
внаслідок відсутності повноти аналізу

if social_service_down:
    social_score = 0.5
    declining_factor += 0.3
```

Кінцева точка /score приймає як запит об’єкт з властивостями applicant_id (ідентифітор кредитної історії): APL-123, features (особливості історії): {}.

У відповіді ми отримуємо:

```
{
  "applicant_id": "APL-123", # ідентифікатор клієнта
  "default_probability": 0.28, # ймовірність дефолту 28%
  "risk_level": "Low", # низький рівень ризику
  "component_breakdown": { # сервісні оцінки
    "ml_score": 0.72, # без дефолту
    "causal_score": 0.65,
    "social_score": 0.80,
```

```
    "weights": {"ml": 0.6, "causal": 0.1, "social": 0.3} #
вплив сервісів на результат, сервіс машинного навчання впливає
найбільше

  },
  "confidence_interval": [0.22, 0.34], # ризик дефолту
варіюється між 22 і 34%
  "explanation": "Low risk applicant with strong traditional
credit and supportive network" # клієнт має тісний зв’язок з
соціальною мережею і має сильну кредитну складову
}
```

Втім, відповідь з серверу може не повністю пояснити для клієнта, чого саме такий результат. За такого сценарію, є кінцева точка /insights, яка приймає ідентичний запит, та повертає відповідь-масив відповідей з сервісів машинного навчання та соціального капіталу:

```
{
  "insights": [
    {
      "category": "traditional_credit",
      "type": "strength",
      "description": "Strong credit profile indicates reliable
repayment history",
      "confidence": "high" # висока впевненість, що кредитна
історія є сильною, зі стабільним сплаченням позики
    },
    {
      "category": "social_capital",
      "type": "strength",
      "description": "Well-positioned in network with
creditworthy connections",
```

```
      "confidence": "high" # з високою вірогідністю багато
соціальних зв’язків зі стабільною кредитною історією
    }
  ]
}
```

Слід також сказати про просунуту кінцеву точку кредитного аналізу за допомогою нейронних агентів. У цій точці беруть участь шість спеціалізованих агентів (аналіз ризиків поведінкових, фінансових, візуальних, причинно-наслідкових, хаі-пояснення, соціального рейтингу). Оркестратор отримує звіти розглянутих агентів, та формує відповідь. Оскільки соціальна точка доповненого кредитного аналізу є порівняно повільнішою, ніж інші кінцеві точки, запит кешується на годину.

Оновлення кешу відбувається коли модель нерезапускається, змінюються ваги особливості клієнтської кредитної історії, або оновлюються дані:

```
@cached("augmented_score", ttl=3600)
async def calculate_score(applicant_id, features):
```

Для уточнення впевненості у результаті, комбінуються неточності з усіх компонентів:

```
# зведення до квадрата робить вплив вагів більш очевидним та
запобігає можливим неточностям зі сторони сервісів
combined_variance = (
    (0.6^2) × ML_variance +
    (0.1^2) × Causal_variance +
    (0.3^2) × Social_variance
)
# інтервал впевненості у прогнозі. 1.96 є відхиленням від
середнього з 95%, константою гаусівського розподілення
confidence_interval = [
    probability - 1.96 × std_dev,
```

```
    probability + 1.96 × std_dev
]
```

Результат буде неточний в залежності від ширини інтервалу впевненості. Сервіс доповненого кредитного аналізу дозволяє визначити різницю в кредитни історіях між різними демографічними групами в залежності від їх соціального становища, зв'язків та підозрілих активностях. Покращене розуміння джерел інформації та чітке пояснення кожного субсервісу всередині (машинне навчання, соціальний капітал, причинно-наслідкові зв'язки). Доповнений кредитний аналіз займає менше часу через паралельне виконання сервісів та кешування операцій.

4. 7. Сервіс прийняття рішень та організації процесу між сервісами

Існування сервісу прийняття рішень є критичним, оскільки машинний кредитний аналіз генерує можливі результати, а бізнес-логіка конвертує їх у готові рішення, які проходять перевірку на зрозумілість звичайними користувачами. Рішення повинні бути прості, по пояснювати суть сервісів.

У нашому випадку, рішення розподілені в чотири категорії: чіткі межі (розпізнання та автовідмова підозрілим кредитним історіям), оцінка ризиків (від A до E), меже кількості (якщо забагато для оцінки, зменшити багатослівність), та кінцеве рішення, яке презентує оцінку з другого етапу, де A/B=дозвіл на позику, C/D=перевірка, E=відмова в кредиті.

Наприклад, A означає, що можливість дефолту менша, ніж 15 відсотків, максимальна позика 50 тисяч доларів, з показником інтересу 0% (боржник сплачує лише позику), позику схвалено; B – 0.15-0.30 (15-30% ризик можливого дефолту), максимальна позика дорівнює 40 тисяч доларів, боржник сплачує два відсотки з боргу, позика також схвалена. Щодо оцінок від C та нижче, позика кредитна історія відправляється на перевірку, ризик дефолту тепер варіюється від 30 до 45-ти відсотків, максимальна позика 30 тисяч доларів, інтерес 4% (боржник платить 4% з заборгованої суми); D – 45%-60% ризик дефолту, 20 тисяч максимальна позика, 6% з суми сплачується додатково, позика на перевірці. E

означає, що у позиці відмовлено (ризик дорівнює 60-ти відсоткам), відповідно без позики.

Інтерес для цієї категорії невизначений, бо немає контексту (самої позики). Деякі випадки потребують додаткової верифікації: перевірка за місцем роботи, проживання, який рівень платні (без врахування додаткових позик). Кінцевою точкою роботи сервісу прийняття рішень є /evaluate, що приймає тіло запиту

```
curl -X POST http://localhost:8003/evaluate \
  -d '{
    # оцінка ризику
    "risk_score": 0.35,
     # сума позики
    "requested_amount": 30000,
     # місячна платня
    "monthly_income": 5000,
     # співвідношення позики до платні
    "debt_ratio": 0.4,
     # тривалість роботи 24 місяці
    "employment_months": 24,
     # нещодавні запити
    "recent_inquiries": 1
  }'
```

Сервіс прийняття рішень досягає необхідного рівня інтересу та умов за межами звичайної оцінки ризиків, завдяки цим полям. Відповідь з цього сервісу є об’єкт з властивостями:

```
{
  # рішення перевірки історії
  "decision": "REVIEW",
  # рівень ризику
  "risk_grade": "C",
```

```
  # умова – перевірка саме рівня платні
  "conditions": ["income_verification"],
  # клієнт за рівнем ризику є середнім та потребує перевірки
  "explanation": "Medium risk applicant requires manual
review",
  # застосовані правила – оцінка ризику, рівень C, фінальним
рішенням є перевірка
  "applied_rules": [
    {"rule_id": "risk_grading", "result": "C"},
    {"rule_id": "final_decision", "result": "REVIEW"}
  ]
}
```

Повний робочий процес має бути структурованим, оскільки сервіси відповідають за свій функціонал у різні проміжки часу – один відповідальний за прогноз кредитних ризиків, другий за прийняття рішень. У таких випадках використовується сервіс окрестрування – цей сервіс координує рух інших сервісів у робочу архітектуру, наприклад: enrich data (сервіс збагачує даними наступний сервіс), за ним credit score (машинне прогнозування отримує дані та повертає кредитний прогноз по клієнтській кредитній історії), після нього fraud check (сервіс перевіряє на шахрайство за допомогою отриманих особливостей кредитної історії), і нарешті policy evaluation (отримує повне тіло запиту з сервісу збагачення для прийняття рішення відмови за дозволу в позиці).

Кінцева точка /score отримує тіло запиту від клієнта. Ця кредитна історія складається з 10-ти особливостей, таких як місячна платня, співвідношення платні до боргу, вік тощо. Якщо особливостей не знайдено, система шукає їх у доступному сервісі кредитного аналізу. Кінцева точка /similar отримує особливості машинного навчання з виконання сервісу кредитного аналізу, без фіксованих значень.

Відповідь складається з особливостей estimated_probability (приблизної ймовірності), risk_assessment (оцінювання ризику), їх оркестратор зберігає у кредитній історії клієнта. Кінцева точка /evaluate:

```
curl -X POST http://localhost:8003/evaluate \
  -d '{
    # числове значення оцінки
    "risk_score": 0.35,
    # максимальна кількість позики
    "requested_amount": 30000,
    # місячна платня
    "monthly_income": 5000,
    # співвідношення боргу до платні
    "debt_ratio": 0.4,
    # місяці працевлаштування
    "employment_months": 24,
    # нещодавні запити
    "recent_inquiries": 1
  }'
```

Додавання нових умов, та більш чітке оцінювання є результатом інтеграції сервісу прийняття рішення з сервісом збагачення даних. HTTP-підключення не можуть бути заблоковані. Щоб запобігти цьому, потрібно використовувти фонові задачі (BackgroundTask):

```
# асинхронна задача, яка приймає запит з кредитною історією, а
також задачами, які мають бути виконані в фоні
@app.post("/apply")
async def apply(req: LoanApplicationRequest, background_tasks:
BackgroundTasks):
    # випадковий зафіксований ідентифікатор
    app_id = str(uuid.uuid4())
```

```
    # кредитні історії з цим ідентифікатором мають мати об’єкт
з ідентифікатором, клієнтські особливості, статус в
очікуванні, запит збільшення позики
    applications[app_id] = {
        "id": app_id,
        "applicant": req.applicant.model_dump(),
        "status": "pending",
        "requested_amount": req.requested_amount,
        "steps": {"enrichment": "pending", "credit_scoring":
"pending",
                  "fraud_check": "pending", "policy":
"pending"},
        "created_at": datetime.utcnow().isoformat(),
    }
    # додати задачу в список фонових задач, з ідентифікатором,
кількістю позики та екземпляром кредитної історії
    background_tasks.add_task(process_application, app_id,
req.applicant, req.requested_amount)
    # як результат, функція повертає ідентифікатор клієнтської
історії, зі статусом в очікуванні
    return {"application_id": app_id, "status": "pending"}
```

Завдяки повертанню результату, клієнти можуть перевірити прогрес обробки власних даних за допомогою кінцевої точки /apply, яка приймає тіло запиту у вигляді ідентифікатору клієнтської кредитної історії, та повертає відповідь з ідентифікатором клієнтської історії, та її статусом.

Повною відповіддю отримання статусу є об’єктом з id:

```
{
  "id": "abc-123",
  # клієнт з іменем і поштою
```

```
  "applicant": {"name": "...", "email": "..."},
  # статус в очікуванні
  "status": "processing",
  # запит стосовно фіксованої позики
  "requested_amount": 30000,
  # кроки, що включають в себе сервіс збагачення (готовий до
повернення результату), кредиту аналізу (готовий), перевірка
шахрайства (в процесі виконання), сервіс прийняття рішень (в
очікуванні на виконання)
  "steps": {
    "enrichment": "complete",
    "credit_scoring": "complete",
    "fraud_check": "running",
    "policy": "pending"
  },
  # час запуску процесу
  "created_at": "2026-04-28T10:00:00",
  # поле буде заповнене, коли процес буде повністю виконаний
  "completed_at": null,
  # аналогічна ситуація з оцінкою згідно виконання всього
процесу
  "score": null,
  # результат перевірки на шахрайство
  "fraud": null,
  # прийняте рішення
  "policy": null,
  # збагачення даних
  "enrichment": null,
  # помилка з’явиться, якщо під час процесу будуть похибки
```

```
  "error": null
}
```

Процес виконання сервісів поділяється на декілька етапів: перебування в очікуванні, перехід до виконання поставленої задачі (опрацювання кредитної історії, етап є найбільш вразливим до отримання похибок), та стан виконання роботи сервісу в процесі, залежно від статусу дозволу, перевірки чи відмови. Для того, щоб імплементувати цей процес, сервіс використовує python менеджер контексту фреймворк FastAPI “lifespan”, без використання обробників подій on_event:

```
@asynccontextmanager
async def lifespan(app: FastAPI):
    # ініціалізація сервісів – збагаченням даними, кредитний аналіз, шахрайство, прийняття рішень
    clients["enrichment"] = ServiceClient(ENRICHMENT_URL)
    clients["scoring"] = ServiceClient(CREDIT_SCORING_URL)
    clients["fraud"] = ServiceClient(FRAUD_DETECTION_URL)
    clients["policy"] = ServiceClient(POLICY_URL)
    # застосування цього ключового слова, як паузи у виконанны ms; вступною логікою, а також виходом з функції, ділення функції навпіл
    yield
    # заверши процес виконання кожної історії
    for client in clients.values():
        await client.close()
    clients.clear()
```

Через поточну збірку, цей процес є лише демонстрацією, через дві причини: процес сервісів зберігається в якості об’єкта-словника, який зникає при перезапуску; також слід сказати, що процес надає несправному сервісу статус error, без перезапуску чи аналізу несправності сервісу. За зберігання посилань на

локальні екземпляри сервісів, а також збереження чутливої інформації відповідає .env файл, всередині якого лежать імена ключів у верхньому регістрі: CREDIT_SCORING_URL=http://localhost:8001; FRAUD_DETECTION_URL=http://localhost:8002; POLICY_URL=http://localhost:8003; ENRICHMENT_URL=http://localhost:8006 тощо.

Після докерної міграції, локальні адреси перезаписані для уникнення конфліктів процесів. Як виглядає процес з урахуванням кількох сервісів на практиці: клієнт передає дані до кінцевої точки /apply (сервіс збагачення даними). З цього сервісу, дані переходять до кредитного аналізу та виявлення шахрайських закономірностей (кілець). Після пройдення цих етапів, кредитна історія переходить до кінцевої точки /evaluate, що є ендпоїнтом сервісу прийняття рішень та бізнес-логіки. З цього сервісу клієнт отримує кінцевий результат, збережений як запис об’єкті-словнику. Сервіс прийняття рішень трансформує кредитні прогнозу у готову бізнес-логіку, тоді як диригент будує процес з залученням кількох сервісів одночасно, за допомогою асинхронних фонових задач.

4. 8. Сервіс збагачення синтетичними даними

Логіка процесів між багатьма сервісами може бути перервана, якщо з самого початку дані не є повними. Кредитний аналіз отримує дані на основі обробки кредитної моделі, яка в свою чергу отримує їх з датасету «Give me some credit». Набір даних є масивом клієнтських кредитних історій, деякі з них не є повними. Заради тестування, вони були зібрані у якості трьох сутностей: клієнтські історії з кредитного бюро, банківські акаунти, та інформація з працевлаштування:

```
ENRICHMENT_SOURCES = {
    # кредитне бюро
    "credit_bureau": {
        "data": [
```

```
            # загальна кількість рядків у клієнтській
кредитній історії
            "total_credit_lines",
            # призначення кредиту
            "credit_utilization",
            # кредитна історія за два роки
            "payment_history_24m",
            # наявність випадків дефолту
            "bankruptcies"
        ]
    },
    # банківський акаунт
    "open_banking": {
        "data": [
            # середній баланс
            "average_balance",
            # вхідні депозити
            "income_deposits",
            # чи повторюються виплати
            "recurring_payments",
            # стабільність акаунту
            "account_stability"
        ]
    },
    # працевлаштування
    "employment": {
        "data": [
            # стан працевлаштування
            "employment_status",
```

```
            # роботодавець перевірений
            "employer_verified",
            # тривалість працевлаштування
            "tenure_months"
        ]
    }
}
```

Оскільки мова йде про демонстрацію роботи додатку, для розробки використовуються тестові дані. У прод будуть інтегровані пакети Equifax (кредитне бюро, що збагачує реальними кредитними звітами і метриками) та Plaid (програмний інтерфейс банківської системи, де http-запити користувачів інтегровані та повертають реальні кредитні дані та власний баланс). Кінцева точка сервісу збагачення синтетичними даними /enrich приймає як тіло запиту об’єкт з властивостями:

```
curl -X POST http://localhost:8006/enrich \
  -d '{
    # ідентифікатор кредитної історії користувача
    "applicant_id": "APL-123",
    # код верифікації особистості клієнта
    "ssn": "123-45-6789",
    # ідентифікатор зв’язку з банківським акаунтом
    "bank_connection_id": "bc_abc123"
  }'
```

Відповіддю /enrich є:

```
{
  "applicant_id": "APL-123",
  # особливості клієнтської історії
  "enriched_features": {
    "RevolvingUtilizationOfUnsecuredLines": 0.35,
```

```
    "NumberOfTimes90DaysLate": 1,
    ...
  },
  # джерела отримання даних
  "data_sources": {
    "bureau": {...},
    "banking": {...},
    "employment": {...}
  },
  # відсоток виконання
  "completeness": 0.85,
  # релевантність даних
  "data_freshness": {
    "bureau": "2024-01-15",
    "banking": "2024-01-16"
  }
}
```

Банківські дані імітуються для того, щоб заповнити проміжки в недостатньо інформативних кредитних історіях:

```
def calculate_banking_score(banking_data):
    indicators = {
        # стабільність платні – стандартне відхилення місячної платні в залежності від середнього значення
        "income_consistency": std(monthly_incomes) / mean(monthly_incomes),

        # відсоток збережень в залежності від платні
        "savings_rate": savings / income,
```

```
        # вчасне внесення боргу в залежності від загального
внесення
        "on_time_rate": on_time_payments / total_payments,

        # вік банківського акаунту
        "account_age": oldest_account_age
    }

    # оцінка в залежності від стабільності акаунту, відсотку
збережень, вчасності сплати та віку акаунту
    score = (
        0.25 * invert(income_consistency) +
        0.25 * savings_rate +
        0.25 * on_time_rate +
        0.25 * normalize(account_age)
    )
    return score
```

Згідно з отриманими даними, не всі даня для збагачення є однаковими – вони поділяються за релевантністю (краща якість «best» в тих, що мають прострочення не більше ніж тиждень). Good для тих, даних, що не старші за місяць. Acceptable, якщо кількість днів не більша за три місяці, та poor коли дані старіші за три місяці. За готовністю – якщо дані є повними на 90%, це краща якість, best. До 70% – good. До 50% дані є обмеженими. Значення менші за половину є недостатніми для якісного прогнозу.

У випадку коли даних недостатньо, допомагає сервіс синтетичних даних. Його роллю є створення синтетичної інформації для розширеного тестування та розробки за умови відсутності реальної інформації. Цей сервіс використовує GAN (генеративні мережі). Приклад роботи: GAN тренується на справжній

кредитній даті, та створює на її основі синтетичну. Кінцева точка /train є часткою цього процесу, саме тренування на справжніх даних:

```
curl -X POST http://localhost:8007/train \
  # файл з даними у .csv форматі
  -F "file=@real_data.csv" \
  # кількість епох тренування
  -F "epochs=300"
```

Відповідь від точки /train:

```
{
  # ідентифікатор моделі
  "model_id": "abc-123",
  # статус натренованості моделі
  "status": "trained",
  # статистика тренування, яка покриває 150000 рядків, з них
11 колонок за допомогою трьохсот епох тренування
  "training_stats": {
    "rows": 150000,
    "columns": 11,
    "epochs": 300
  }
}
```

Кінцева точка /generate створює синтетичні вибірки з кредитної дати, посилаючи тіло запиту:

```
curl -X POST http://localhost:8007/generate \
  -d '{
    "model_id": "abc-123",
    # кількість даних у вибірці
    "n_samples": 10000,
    # за якої умови дані для вибірки мають бути відібрані
```

```
    "conditions": {"risk_level": "high"}
  }'
```

Для тестування можуть бути відтворені сценарії за умовою:

```
# значення високого ризику
high_risk = generator.generate(
    # генерація 1000 екземплярів вибірки, з умовою що дані
місячну платні на рівні від 2000 до 4000
    n_samples=1000,
    # використання кредиту, який вже доступний, gte більше або
дорівнює 80 відсоткам
    conditions={
        "MonthlyIncome": {"between": [2000, 4000]},
        "RevolvingUtilizationOfUnsecuredLines": {"gte": 0.8}
    }
)

# генератор синтетичних даних для виявлення шахрайства, з
ключовою особливістю, що кількість 90-денного просрочення
сплати більше, ніж 5 разів
fraud = generator.generate(
    n_samples=100,
    conditions={"NumberOfTimes90DaysLate": {"gte": 5}}
)
```

За цю роботу відповідає функція validate_quality:

```
def validate_quality(synthetic, real):
    report = {
        # порівняння колонок за подібністю
        "column_similarity": compare_distributions(),
```

```
        # різниця матриці кореляцій
        "correlation_matrix_diff": compare_correlations(),

        # всі історії мають бути унікальними
        "no_exact_matches": check_no_duplicates(),
         # підрахунок оцінки якості
        "quality_score": calculate_overall()
    }
    return report
```

де цільовою точкою є більше ніж 80% схожості між кредитними історіями, але без існування дублікатів, та з 90-відсотковим збереженням кореляції.

Оскільки ctgan-модель тренується на реальних клієнтських даних (з датасету у нашій демонстрації), приватність є необхідністю. Для збільшення приватності в модель додається більше шуму, це не дозволяє моделі зберегти в пам’яті конкретного клієнта. Досягається це за допомогою функції generate_with_privacy(real_data, epsilon=1.0):

```
def generate_with_privacy(real_data, epsilon=1.0):
    # статистичний аналіз наївного байєса, який дозволяє зберегти контекст конкретної особливості, унеможливуючи знаходження конкретного клієнта, за допомогою параметра, який відповідає за рівень шуму для збереження приватності
    model = dp.GaussianNB(epsilon=epsilon)
    model.fit(real_data)

    # генерація зі збереженням приватності
    return model.sample(n_samples)
```

Після отримання шуму, приватні дані потрапляють до кінцевої точки диригента:

```
async def process_application(app_id, data):
```

```
    # досягнення кінцевої точки збагачення даними
    enriched = await enrichment_service.post("/enrich", json={
        "applicant_id": app_id,
       # дані клієнта
        "ssn": data.ssn
    })

    # досягнення кінцевої точки кредитного аналізу з
додаванням отриманих особливостей кредитної історії клієнта
    score = await
credit_service.predict(enriched.enriched_features)
```

Як виглядає залучення ctgan у процес створення синтетичних даних в розробці:

```
def train_model_development():
    # генерація синтетичних даних
    synthetic = synthetic_service.generate(n_samples=100000)

    # тренування моделі
    model = train_credit_model(synthetic)

    # валідація на основі невеликої вибірки
    real = load_real_data(limit=1000)
    metrics = evaluate_model(model, real)
    # функція повертає модель, натреновану на квазі-реальних
синтетичних даних, та метрики на основі реальної інформації
від клієнтської кредитної історії
    return model, metrics
```

4. 9. Загальний сервіс, пакети та інфраструктура

Щоб уникнути копіювання http-клієнтів, моделей та конфігурації між мікросервісами, застосовуються загальні пакети. Як це працює: окрім папок з індивідуальними сервісами існує окрема папка для загального сервісу, до якого мають доступ всі індивідуальні сервіси: services/shared/credscore_shared: models.py (валідація моделей через pydantic), client.py (http клієнт), config.py (конфігурація), events.py (обробники подій). Сервіс досягають загальний сервіс за допомогою команди pip (менеджер пакетів) install services/shared (встановлення за директорією).

Загальний сервіс (як і його конфігурація) часто є більшим за обсягом, ніж індивідуальні сервіси. Це через те, що загальний сервіс є набором інструментів, що не належать конкретному сервісу, використовуються багато разів у різних точках. Pydantic-валідація моделей:

```
class ClientData(BaseModel):
    """The 10 features credit models need."""
    RevolvingUtilizationOfUnsecuredLines: float = Field(..., 
ge=0, le=1)
     # ge – більш ніж, le – менш ніж
    age: int = Field(..., ge=18, le=100)
    DebtRatio: float = Field(..., ge=0)
    MonthlyIncome: float = Field(..., ge=0)
    # ...

# типізована відповідь від сервісу кредитного аналізу
class PredictionResponse(BaseModel):
    prediction: bool
    default_probability: float = Field(..., ge=0, le=1)
    risk_level: str
```

```
    # типізований масив, який складається з інтерфейсу
факторів ризику
    risk_factors: List[RiskFactor]

# аналогічна типізована відповідь від кінцевої точки сервісу,
яка перевіряє його здоров’я
class HealthCheckResponse(BaseModel):
    # кортеж з трьох фіксованих значень
    status: Literal['healthy', 'degraded', 'unhealthy']
    service: str
    timestamp: datetime
    # секунди роботи з плаваючою точкою
    uptime_seconds: float
```

Плюсами в типізації є стабільність роботи без необхідності перевіряти тип, автозаповнення, та валідація типів.

Ще одним інструментом у загальному сервісі є HTTP-клієнт, який діє як шаблон для використання іншими сервісами:

```
class ServiceClient:
    def __init__(self, base_url: str, timeout: float = 30.0):
         # адреса сервісу, виконання його 30 секунд
        self.base_url = base_url
        self.client = httpx.AsyncClient(timeout=timeout)

     # запит для отримання даних
    async def get(self, path: str) -> Dict:
        try:
            # динамічне отримання відповіді в залежності від
кінцевої точки
```

```
            response = await
self.client.get(f"{self.base_url}/{path}")
            # отримання статусу сервісу
            response.raise_for_status()
            # отримання відповіді у json-форматі
            return response.json()
        # відловлення помилок за допомогою структури
try/except. Всередині try блоку розміщена основна логка. Якщо
під час роботи є помилка, спрацьовує блок except, всередині
якого повертається помилка в залежності від її типу
        except httpx.ConnectError:
            raise ServiceError("Service unreachable")
        except httpx.TimeoutException:
            raise ServiceError("Service timeout")
    # пост метод для надсилання тіла запиту, кредитної історії
у нашому випадку
    async def post(self, path: str, json: Dict) -> Dict:
        ...
    # закриття доступу до сервісу
    async def close(self):
        await self.client.aclose()
```

Індивідуальні сервіси зі свого боку імпортують шаблон клієнта from credscore_shared.client:

```
from credscore_shared.client import ServiceClient

# створення клієнту
credit_client = ServiceClient(os.getenv("CREDIT_SCORING_URL"))
```

```
# використання клієнту для досягнення пост методу та
надсилання тіла запиту у json-форматі.
result = await credit_client.post("/predict",
json={"client_data": data})
```

Основними особливостями клієнтського шаблону є пулінгування з'єднань (створення пулу з'єднань, та його використання за потреби. Якщо всі з'єднання зайняті, очікування, поки з'єднання не повернеться назад до пулу) та помилки в залежності від типу індивідуального сервісу. Також використовується загальна конфігурація, еластична в залежно від конкретного сервісу. Class ServiceConfig(BaseSettings): (базові значення версії, порту, хосту, посилання на базу даних для кешування Redis, а також на інші адреси індивідуальні сервіси):

```
class ServiceConfig(BaseSettings):
    # інформація про сервіс
    SERVICE_NAME: str
    SERVICE_VERSION: str = "1.0.0"

    # мережа
    PORT: int = 8000
    HOST: str = "0.0.0.0"

    # залежності, посилання на базу даних редіс
    REDIS_URL: str = "redis://localhost:6379"

    # доступні сервіси
    CREDIT_SCORING_URL: str = "http://localhost:8001"
    FRAUD_DETECTION_URL: str = "http://localhost:8002"

    # особливості сервісу, можливість збереження в кеші, так
по замовчуванню
```

```
    ENABLE_CACHING: bool = True

    class Config:
        env_prefix = ""
```

Наприклад, from credscroe_shared.config:

```
from credscore_shared.config import ServiceConfig

config = ServiceConfig()
print(f"Запускаю {config.SERVICE_NAME} за адресою порту
{config.PORT}")
# Середовище розробки
export CREDIT_SCORING_URL=http://localhost:8001
export LOG_LEVEL=DEBUG

# Прод
export CREDIT_SCORING_URL=http://credit-scoring:8000
export LOG_LEVEL=INFO
```

Під час розробки використовуються два типи змінних середовища (env) – development та production. Розробка посилається на локальні адреси, в той час як прод звертається до реальних, зареєстрованих доменів, наприклад export CREDIT_SCORING_URL_DEV = http://localhost:8001 (розробка), export CREDIT_SCORING_URL_PROD = http://credit-scoring:8000.

Для того, щоб індивідуальні сервіси могли зв’язатись між собою, використовується потокова платформа Kafka:

```
class EventPublisher:
    def __init__(self, kafka_servers: str):
        self.producer = KafkaProducer(
            # використання серверів кафка в якості брокерів
між сервісами
```

```
            bootstrap_servers=kafka_servers,
            # серіалізація запитів та відповідей у одну
стрічку для легкості парсингу
            value_serializer=lambda v:
json.dumps(v).encode('utf-8')
        )
    # зв’язок між сервісами за допомогою надсилання події з
метаданими до індивідуального сервісу
    def publish(self, topic: str, event: Dict):
        event_with_meta = {
            # ** є символом спред-операції – всі властивості
об’єкта-словника з додаванням додаткових метаданих timestamp
event id
            **event,
            # ідентифікатор події
            "timestamp": datetime.utcnow().isoformat(),
            "event_id": str(uuid.uuid4())
        }
         # продюсер надсилає подію до сервісу
        self.producer.send(topic, event_with_meta)
```

Структура події (об’єкта-словника Dict) складається з

```
class CreditEvent(BaseModel):
    event_type: str
    applicant_id: str
    application_id: str
    timestamp: datetime
    data: Dict
```

Тригерами надсилання до індивідуальних сервісів є тематики: “credit.applications” (сервіс кредитного аналізу), “credit.decisions” (сервіс

прийняття рішень), “credit.scores” (створення синтетичних даних), “fraud.alerts” (підозрілі закономірності). Зберігання в короткочасній пам’яті відбувається завдяки базі даних Redis:

```
import redis
# хостом адрес бази даних є редіс за портом 6379
cache = redis.Redis(host='redis', port=6379)

# збереження в короткочасній пам’яті ресурсні операції: аналіз історії користувача, тимчасове збереження в базі даних, перш ніж дані оновляться сервісом
@cached("credit_score", ttl=1800)
async def get_credit_score(applicant_id):
    return expensive_calculation()
```

Аналізуючи загальну інфраструктуру, слід також сказати про контейнеризацію Docker.

Кожний мікросервіс додатку, включно з клієнтом та шлюзом API огорнутий в ізольований контейнер. Кожен з цих контейнерів займає своє місце у кластері контейнерів. Кластер створюється за допомогою yaml-конфігурації docker-compose, і має у собі кожен процес у додатку. Кластер утворюється за образу та початкової конфігурації всередині файлу Dockerfile:

```
# копія загального сервісу до директорії всередині docker контейнеру
COPY services/shared/credscore_shared /app/shared/credscore_shared
# копія локальної конфігурації
COPY services/shared/pyproject.toml /app/shared/pyproject.toml
# запуск загального сервісу всередині контейнеру
RUN pip install -e /app/shared
```

```
# копія коду сервісу кредитного аналізу
COPY credit_scoring/app /app/app
```

Кластер контейнерів можна запустити в будь-якому середовищі, оскільки всередині контейнеру Docker має все що є – змінні середовища, пакети, конфігурації та операційну систему. Для того, щоб запустити docker-контейнер, користувач має мати лише запущений двигун docker desktop.

ВИСНОВКИ

У рамках даної кваліфікаційної роботи було успішно виконано проектування, комплексну програмну реалізацію та розгортання інтелектуальної системи аналізу кредитних ризиків на основі машинного навчання. Поставлену дослідницьку та інженерну мету проєкту було повністю досягнуто, що дозволило створити цілісну, масштабовану та інтерпретовану екосистему для фінансового моніторингу.

Під час виконання проєкту було детально проаналізовано сучасні теоретичні та практичні методики кредитного аналізу, передові алгоритми виявлення шахрайства, а також технології генерації синтетичних даних для збагачення обмежених вибірок. Практична побудова прогностичного ядра системи базувалася на розробці та навчанні інтелектуальної моделі на основі реального спеціалізованого набору даних (датасету) «Give Me Some Credit». Як ключовий обчислювальний інструмент було залучено прогресивну бібліотеку градієнтного бустингу LightGBM, яка продемонструвала високу точність класифікації та швидкодію при обробці багатовимірних фінансових метрик. З метою подолання проблеми «чорного ящика», характерної для ансамблевих моделей, та забезпечення повної прозорості прийнятих рішень, натреновану модель було огорнуто детальними поясненнями від математичної бібліотеки SHAP (Shapley Additive exPlanations). Це дозволило декомпозувати логіку роботи алгоритмів та наочно продемонструвати локальний і глобальний валідаційний внесок кожної окремої характеристики позичальника у кінцевий результат оцінювання ризику.

Важливим технологічним досягненням роботи є розробка та впровадження доповненого кредитного аналізу, обчисленого на основі синергетичної взаємодії окремих автономних сервісів кредитного аналізу, виявлення шахрайства та автоматизованого прийняття рішень. У ході інженерної реалізації системи було продемонстровано практичне використання складних кредитних методологій, суворе об'єктно-орієнтоване проектування із застосуванням ізольованих

модульних класів та пакетів, а також налагодження надійних внутрішніх протоколів зв’язку між сервісами. Задля суттєвого підвищення швидкодії та оптимізації мережевих ресурсів розподіленої інфраструктури було розроблено та впроваджено архітектурний механізм збереження одноманітних відповідей з сервісів у короткочасній пам’яті (кеші), що дозволило уникнути надлишкових повторних обчислень. Уся створена мікросервісна система була розгорнута та запущена у гнучкому контейнерному середовищі Docker, що гарантує її ізоляцію та кросплатформенну стабільність.

Окреме місце в архітектурі посідає організація інтелектуального кредитного аналізу за допомогою оркестру автономних агентів, де кожен агент виконує чітко делеговану йому роль, включаючи первинне отримання даних, безпосередній кредитний аналіз, автоматичне створення фінансових звітів та динамічне формування підсумкової відповіді на основі виявлених підозрілих закономірностей у поведінці користувачів. Крім того, було виконано значний обсяг роботи над спеціалізованим сервісом соціального капіталу, який дозволяє ефективно відстежувати приховані зв’язки між клієнтськими кредитними історіями, детально аналізувати їхню поточну фінансову репутацію та враховувати вік позичальників. Усі ці процеси взаємодії між сервісами були чітко скоординовані як у межах внутрішньої мікросервісної архітектури, так і у зовнішній взаємодії з клієнт-серверною архітектурою всього додатку.

З боку користувацького інтерфейсу клієнтом у реальному часі отримується комплексна структурована відповідь від аналітичних сервісів, яка миттєво трансформується у наочні та зручні для сприйняття візуальні графіки. Програмний каркас фронтенд-частини побудовано на основі передових бібліотек React та фреймворку Next.js. Для стилізації інтерфейсу було сформовано сучасну комбіновану структуру, що поєднує можливості препроцесорів SCSS та лінійної (утилітарної) стилізації Tailwind CSS безпосередньо всередині компонентів системи. Для створення сучасного, динамічного та естетичного дизайну веб-додатку було використано базові інтерактивні візуальні компоненти з бібліотеки AceternityUI.

Створений програмний продукт має чітку логіку навігації та складається з трьох ключових сторінок, до яких увійшли оглядова сторінка (головна панель моніторингу), сторінка з технічною документацією та розгорнута сторінка метрик. Основну роботу в межах інтерфейсу було сконцентровано навколо проектування сторінки метрик сервісів, всередині якої було створено інтерактивну систему вкладок для відображення детального кредитного аналізу, швидкого пошуку за подібними кредитними історіями, безпосереднього запуску генерації синтетичних даних, аналізу даних дрифту (зсуву розподілу даних), моніторингу метрик за групами соціального капіталу, а також виведення результатів агентського кредитного аналізу.

Поточний проєкт повністю сконцентрований на бездоганній реалізації головного обчислювального та аналітичного функціоналу. Надалі для переведення системи у статус готового комерційного продукту планується розширити інфраструктуру безпеки та впровадити повноцінну авторизацію користувачів через готові хмарні платформи Supabase або Pocketbase, налаштувати безпечні перехресні браузерні запити (CORS) для взаємодії з іншими доменами, а також повністю перевести весь мережевий обмін даними додатку на захищений шифрований протокол Hyper Text Transfer Protocol Secure (HTTPS). Розроблена система демонструє високу інженерну зрілість та є перспективним інструментом для автоматизації процесів оцінювання фінансових ризиків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Thomas Kwa. Measuring AI Ability to Complete Long Tasks. URL: https://metr.org/blog/2026-03-19-measuring-ai-ability-to-complete-long-tasks/?utm_source=www.therundown.ai&utm_medium=newsletter&utm_campaign=ai-s-moore-s-law-emerges&_bhlid=8ff10f37396291f5cb09a5431c0a47b50c620201 (дата звернення: 13.02.2026).

2. Amazon. Human-in-the-Loop. URL: https://awslabs.github.io/generative-ai-atlas/topics/3_0_architecture_and_design_patterns/3_1_system_and_application_design_patterns_for_genai/3_1_1_foundation_architecture_components/3_1_1_8_additional_components/3_1_1_8_1_human_in_the_loop/3_1_1_8_1_human_in_the_loop.html (дата звернення: 14.02.2026).

3. Kosinski M. Black Box. URL: https://www.ibm.com/think/topics/black-box-ai (дата звернення: 15.02.2026).

4. Silent Eight. OFAC Compliance. URL: https://www.silenteight.com/blog/ai-and-machine-learning-in-sanctions-screening-where-ofac-draws-the-line (дата звернення: 18.02.2026)

5. Interfax-Україна. Використання ШІ в банках обмежено вимогами НБУ та GDPR – експерт. URL: https://media.interfax.com.ua/media/thumbs/images/2017/10/a9W3nlEl9BHc.jpg (дата звернення: 18.02.2026).

6. Rao A., Keller T. Explainable AI Frameword Implementation, p.6. URL: https://ecsenet.com/index.php/2576-6821/article/view/709/281 (дата звернення: 19.02.2026).

7. Bahlool R. Performance, Fairness, and Explainability in AI-Based Credit Scoring: A Systematic Literature Review. URL: https://www.mdpi.com/1911-8074/19/2/104 (дата звернення: 21.02.2026).

8. Lumenova.ai. Group vs. Individual Fairness in AI. URL: https://www.lumenova.ai/blog/group-fairness-vs-individual-fairness/ (дата звернення: 26.02.2026).

9. Ukraine Business News. During the war in Ukraine, an alternative to the state eOselya initiative has developed: In 2026, developer installments are expected to fund 65-90% of new housing transactions. URL: https://ubn.news/during-the-war-in-ukraine-an-alternative-to-the-state-eoselya-initiative-has-developed-in-2026-developer-installments-are-expected-to-fund-65-90-of-new-housing-transactions/ (дата звернення: 26.02.2026).

10. Sebastian D. Ethical Architecture in AI-Driven Credit: Balancing Inclusion, Fairness, and Transparency. URL: https://jisem-journal.com/index.php/journal/article/view/13275 (дата звернення: 28.02.2026).

11. Vilarino R., Vicente R. An experiment on the mechanisms of racial bias in ML-based credit scoring in Brazil. URL: https://ar5iv.labs.arxiv.org/html/2011.09865#Sx5.F18 (дата звернення: 28.02.2026).

12. Schmidt J-H. Elevating Developers' Accountability Awareness in AI Systems Development, p.130-131. URL: https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1819&context=bise (дата звернення: 28.02.2026).

13. Barclays. What is credit scoring? URL: https://www.barclays.co.uk/help/credit-rating/credit-scoring/#back=%2Fcontent%2Fbarclaysuk%2Fen%2Fhelp%2Fcategories%2Fabout-you%2Fyour-credit-rating.html%3Fdirect%3Dtrue%26offset%3D20%26origin%3Dhelp.barclays.co.uk%26q%3Dcredit%252brating%26spellcheck%3Dfalse (дата звернення: 01.03.2026).

14. ICO. Credit. URL: https://ico.org.uk/for-the-public/credit/#:~:text=Credit%20reference%20agencies%20(CRAs)%20give,are%20Equifax%2C%20Experian%20and%20TransUnion. (дата звернення: 01.03.2026).

15. Ballegeer, M., Bogaert, M., & Benoit, D. F. (2026). Evaluating the stability of model explanations in instance-dependent cost-sensitive credit

scoring. *European Journal of Operational Research*. URL: https://www.sciencedirect.com/science/article/abs/pii/S0377221725004230 (дата звернення: 01.03.2026).

16. Claude AI. Sonnet 4.6. URL: https://claude.ai/settings/general (дата звернення: 05.03.2026).

17. Bernal L., Rastelli G., Pinzi L. Improving Machine Learning Classification Predictions through SHAP and Features Analysis Interpretation, p.9. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC12606625/pdf/ci5c02015.pdf (дата звернення: 07.03.2026).

18. AMD. LightGBM Documentation. URL: https://rocm.docs.amd.com/projects/lightgbm/en/docs-25.11/index.html (дата звернення: 08.03.2026).

19. AMD. What is ROCm? URL: https://rocm.docs.amd.com/en/latest/what-is-rocm.html (дата звернення: 09.03.2026).

20. Huo P., ryanzhang1230, Glogowski D. NVIDIA-AI-Blueprints: quantitative-portfolio-optimization. URL: https://github.com/NVIDIA-AI-Blueprints/quantitative-portfolio-optimization (дата звернення: 09.03.2026).

21. Amit H. One-Hot Encoding Explained. URL: https://medium.com/@heyamit10/one-hot-encoding-explained-0b0130ccd78e (дата звернення: 21.03.2026).

22. CSDN. A new paradigm for financial risk control: building a millisecond credit assessment and fraud detection system with FAISS. URL: https://blog.csdn.net/gitblog_00071/article/details/151504737 (дата звернення: 28.03.2026).

23. Mehrotra A. Building Multi-Agent Systems for Financial Document Analysis with LangGraph. URL: https://www.linkedin.com/posts/anujamehrotra_github-anujamehrotrafinancial-agent-activity-7357747927831728128-D5NF/ (дата звернення: 28.03.2026).

24. Chidiebere V.-Ch. Building a Production-Grade Credit Risk System. URL: https://medium.com/@chidiebere.vincent/building-a-production-grade-credit-risk-system-4c22f94f2c83 (дата звернення: 04.04.2026).

25. Barcklow D., Dulepet P. Machine Learning in Finance: Comprehensive Guide. URL: https://www.capitalone.com/tech/machine-learning/similarity-search-graph-embeddings/ (дата звернення: 05.04.2026).

26. Kennedy W. B. Shared Nearest Neighbours: A More Robust Distance Metric. URL: https://towardsdatascience.com/shared-nearest-neighbors-a-more-robust-distance-metric-064d7f99ffb7/ (дата звернення: 06.04.2026).

27. Wang B., Nunthasen K., Sinnarong N., Authariyapanitkul K. Enhancing Credit Risk Assessment in Germany: A GAN-Based Approach with Forward-Looking Variables. URL: https://so07.tci-thaijo.org/index.php/IJSASR/article/view/6969/5495 (дата звернення: 07.04.2026).

28. Mendez C. Introduction to Causal Inference: Double Machine Learning. URL: https://carlos-mendez.org/post/python_doubleml/ (дата звернення: 09.04.2026).

29. Sifuna F. Causal Inference in Credit Scoring: Unlocking Deeper Insights with Social Capital Data. URL: https://www.linkedin.com/pulse/causal-inference-credit-scoring-unlocking-deeper-insights-sifuna-9fecf/ (дата звернення: 10.04.2026).

30. Shi Yu., Yang Zhi., Wang Yi., Zhou L. CreditXAI: A Multi-Agent System for Explainable Corporate Credit Rating. School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, China. URL: https://browse-export.arxiv.org/pdf/2510.22222 (дата звернення: 11.04.2026).

31. Ouyang Sh., Bai Q. Feng H., Hu B. Bitcoin Money Laundering Detection via Subgraph Contrastive Learning. URL: https://www.mdpi.com/1099-4300/26/3/211/xml (дата звернення: 13.04.2026)

32. S. Nayak. Calibrated Credit Intelligence: Shift-Robust and Fair Risk Scoring with Bayesian Uncertainty and Gradient Boosting. URL: https://arxiv.org/html/2603.06733v1 (дата звернення: 14.04.2026).

33. Shamsi K., Novokmet D., Peters J., Liu M. L., Edwards P. K., Khoshdel V. LendNova: Towards Automated Credit Risk Assessment with Language Models. URL: https://arxiv.org/html/2601.02573v1 (дата звернення: 18.04.2026).

ДОДАТКИ

А1 «HITL Патерн дерева рішень»

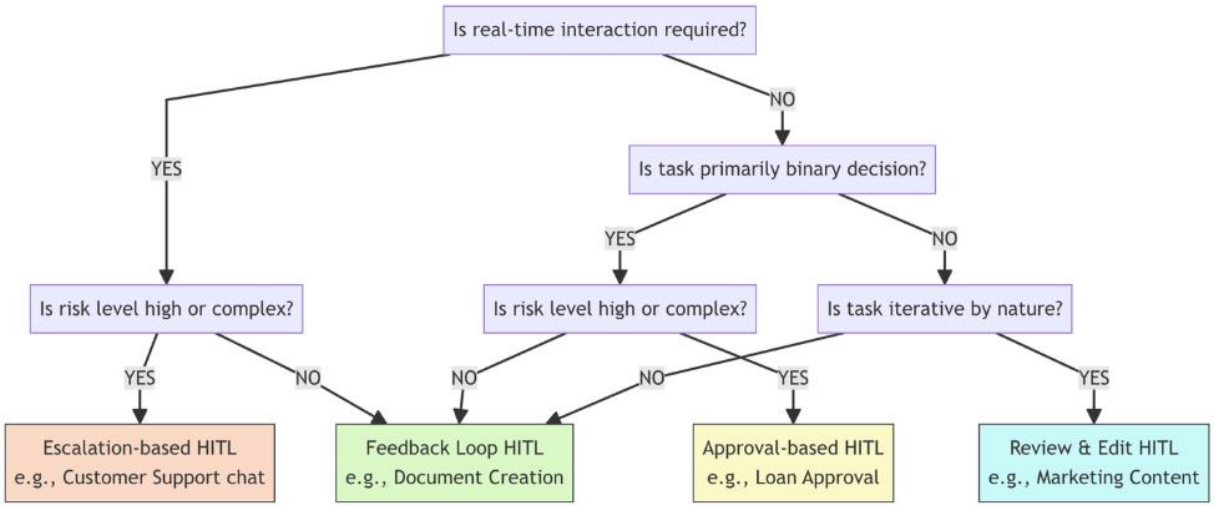
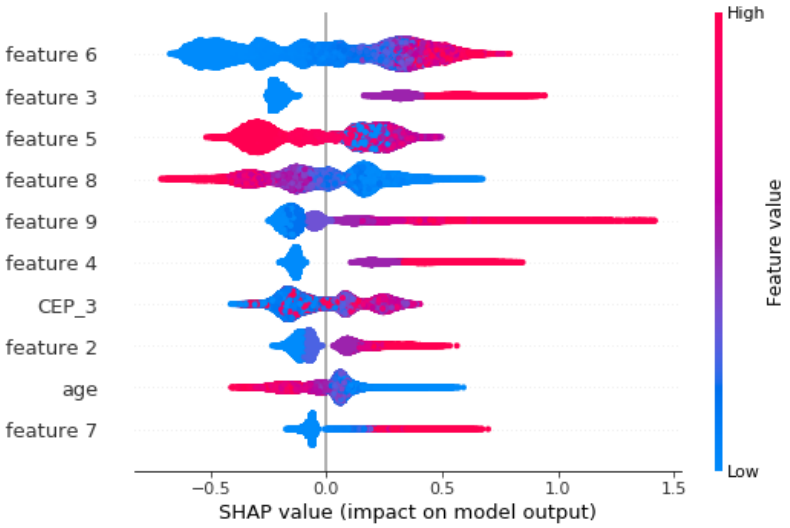


Figure 1: HITL Pattern Selection Decision Tree

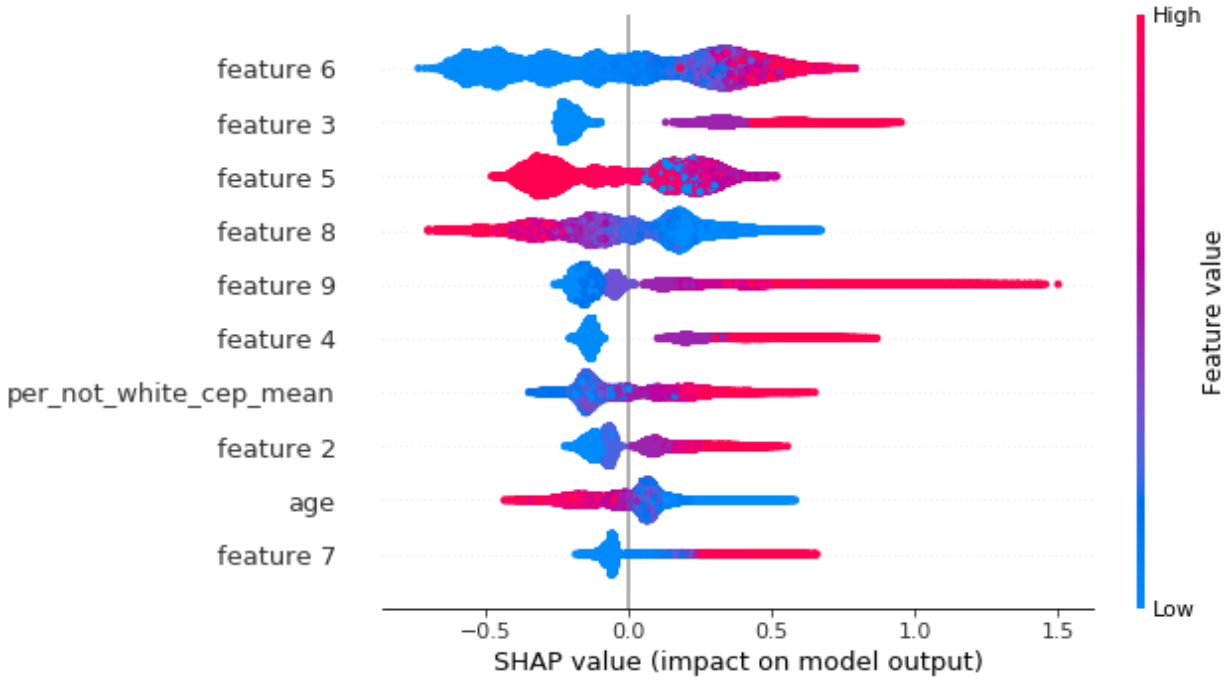
А2 «SHAP Формула Середнього всіх комбінацій індивідуальних ознак»

$$\phi_i(f, \mathbf{x}) = \frac{1}{|N|} \sum_{S \subseteq N \setminus \{i\}} \frac{\left[f(\mathbf{x}_{S \cup \{i\}}) - f(\mathbf{x}_S) \right]}{\binom{|N|-1}{|S|}}$$

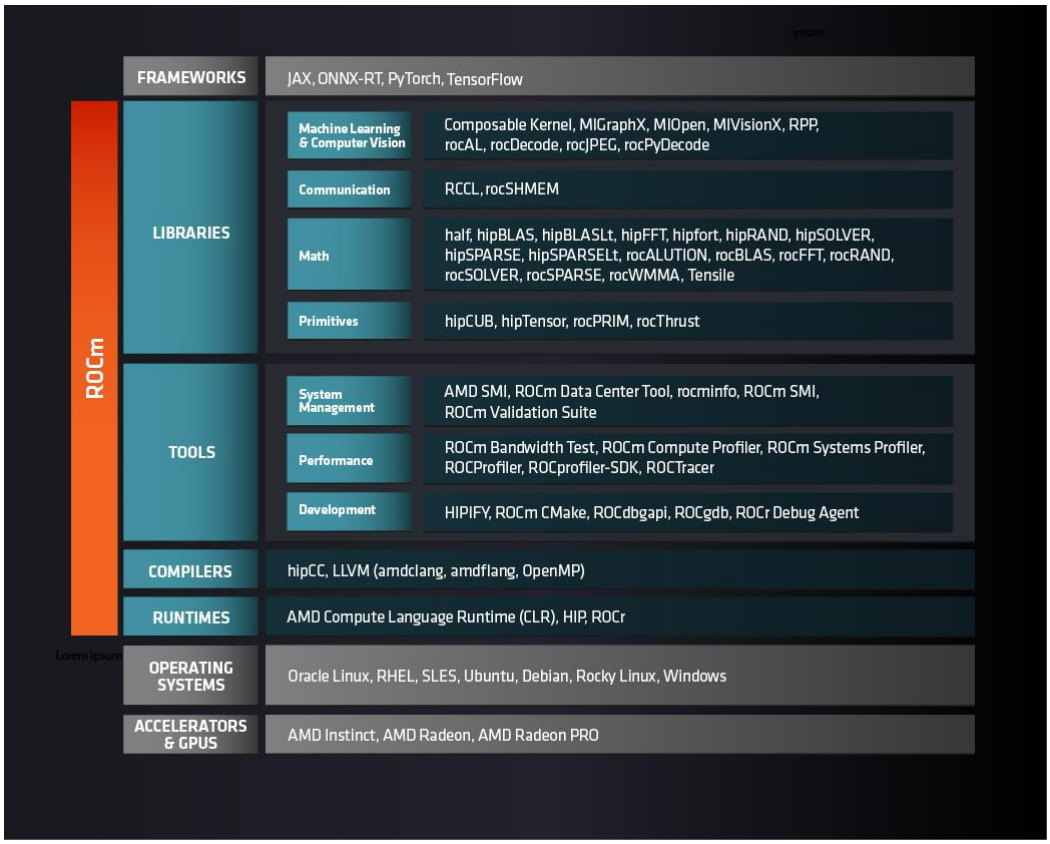
А3 «CEP-3 упередженість»



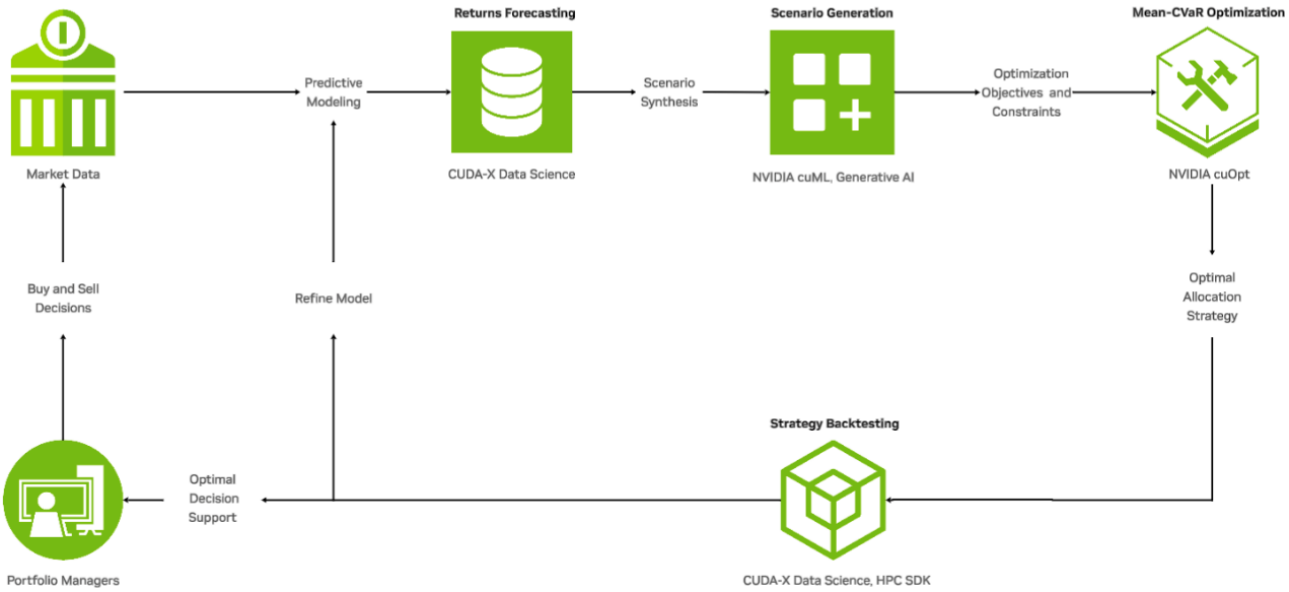
А4 «порівняння CEP-3 з «не білим» поштовим індексом»



А5 «фреймворки ROCm»



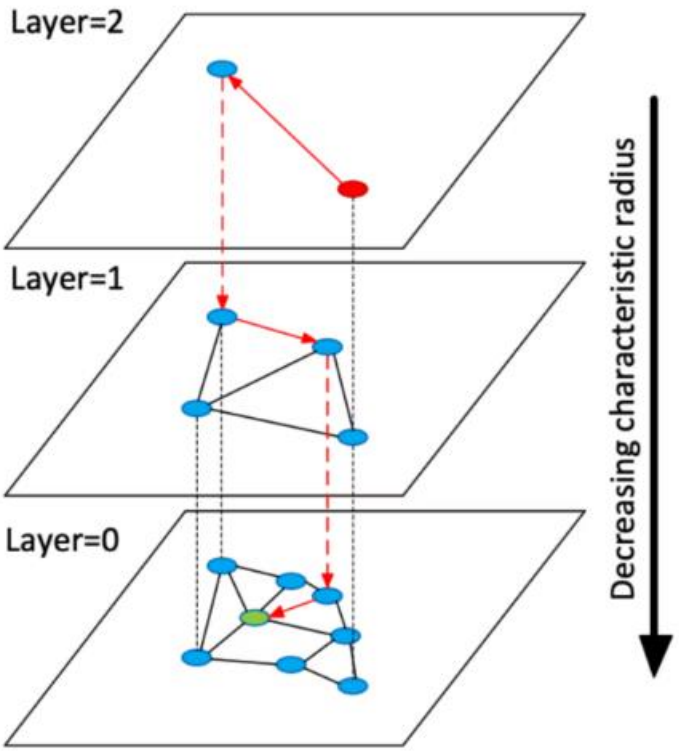
А6 «оптимізація фінансової інформації за допомогою NVIDIA»



А7 «one-hot encoding (кодування один до одного)»

Color	Red	Blue	Green
Red	1	0	0
Blue	0	1	0
Green	0	0	1

А8 Hierarchical Navigable Small World (HNSW)



А9 Модель аналізу кредитних ризиків для GAN (Generative Adversarial Networks)

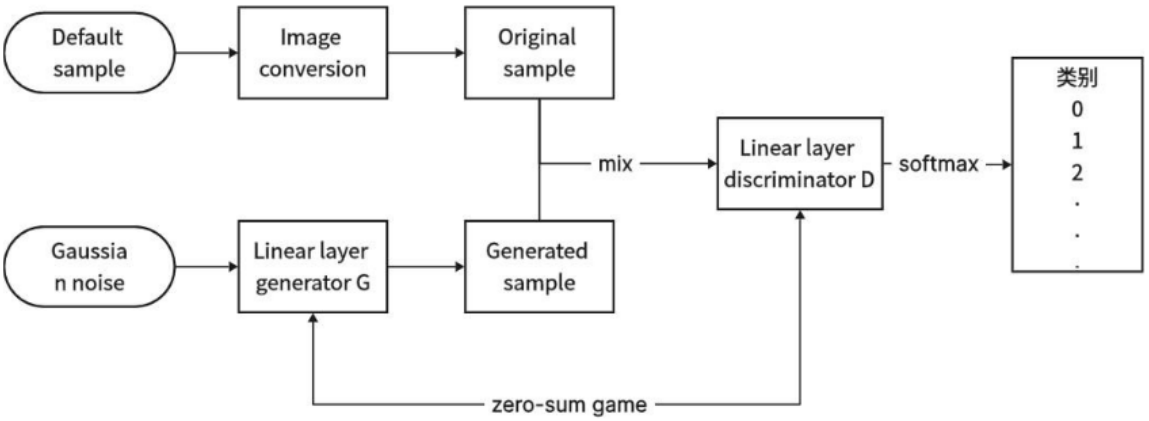
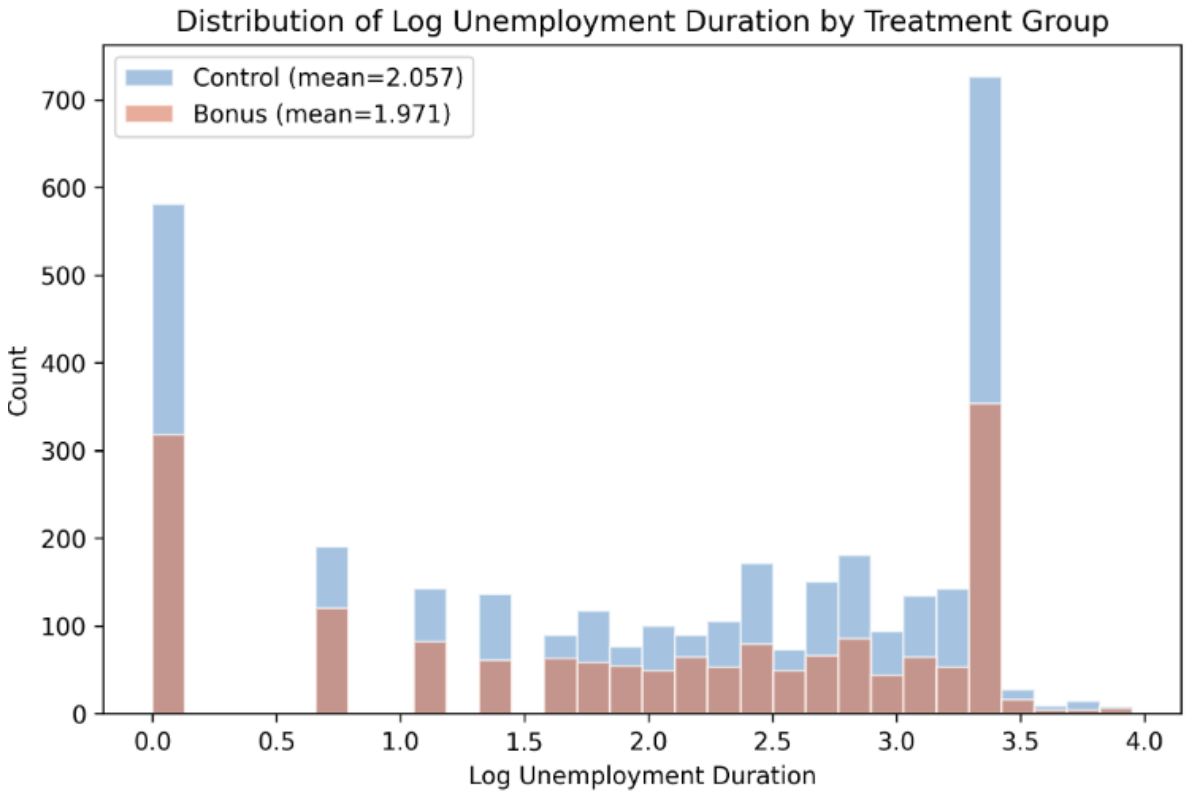
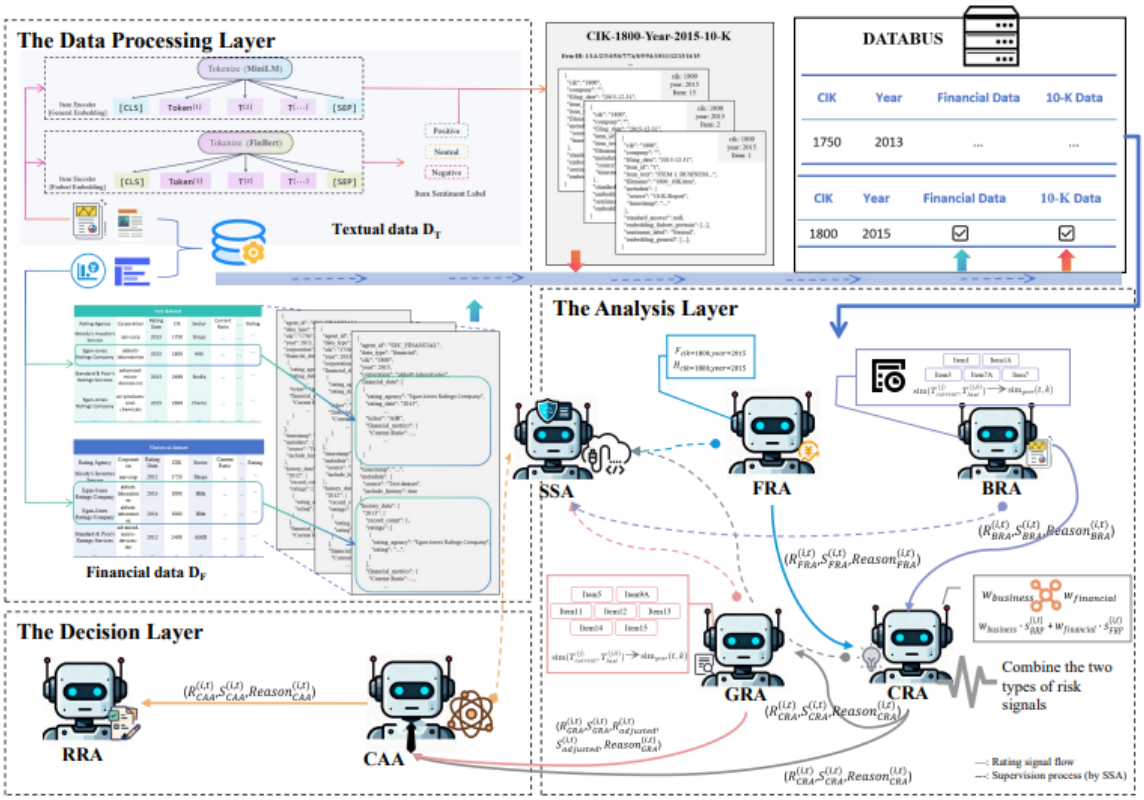


Figure 2 The credit scoring model of GAN

А10 «Порівняння контрольної та бонусної групи з визначенням причинного ефекту»



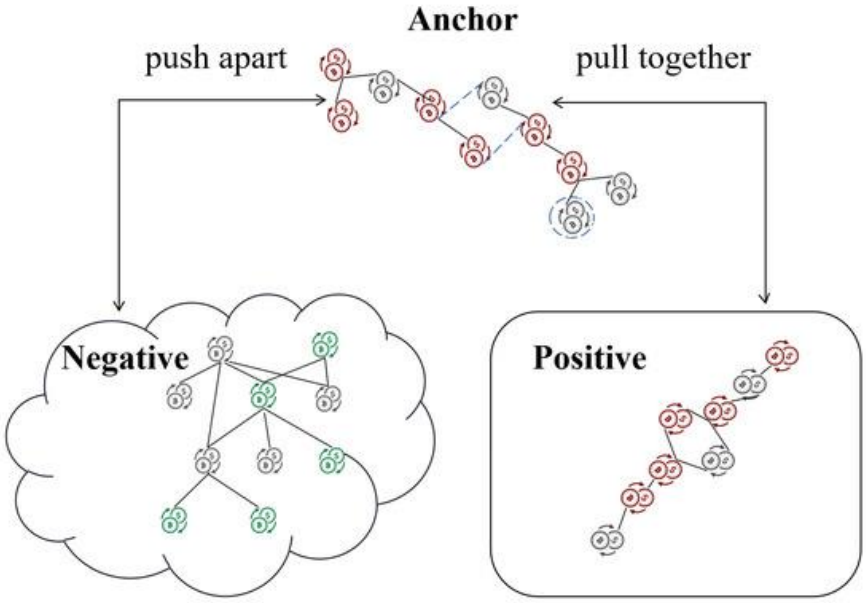
Додаток А11 «CreditXAI, чотири рівня і сім агентів»



А12 «Формула контрастного навчання»

$$L_{\mathrm{CL}} = -\sum_{m=1}^{M} \log \left\{ \frac{1}{|\mathcal{P}(m)|} \times \sum_{p \in \mathcal{P}(m)} \frac{\exp\left(\mathrm{sim}\left(\mathbf{g}_m, \mathbf{g}_p\right)\tau\right)}{\sum_{n \in \mathcal{N}(m)} \exp\left(\mathrm{sim}\left(\mathbf{g}_m, \mathbf{g}_n\right)\tau\right)} \right\}.$$

А13 «Кероване контрастне навчання для побудови графів»



Практична імплементація виконана всередині репозиторію на Github:

https://github.com/Hakkarian/CredsCore