

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

**Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу**

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«__» _____ 2026 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

**на тему: «Моделі поведінкової аналітики і машинного навчання для
конструювання ознак і прогнозування конверсії користувачів
мобільного застосунку»**

Виконала:

студентка IV курсу, групи КА-22
Гаврилюк Богдана Олександрівна

Керівник:

професор кафедри ММСА, д.т.н., проф.
Недашківська Надія Іванівна

Консультант з економічного розділу:

доцент кафедри ЕК, к.е.н., доц.
Рощина Надія Василівна

Консультант з нормоконтролю:

доцент кафедри ММСА, к.ф.-м.н.
Статкевич Віталій Михайлович

Рецензент:

д.т.н., проф. кафедри ІІІ, проф.
Данилов Валерій Якович

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

Студентка _____

Київ – 2026 року

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

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«__» _____ 2026 р.

ЗАВДАННЯ
на дипломну роботу студентці
Гаврилюк Богдані Олександрівні

1. Тема роботи «Моделі поведінкової аналітики і машинного навчання для конструювання ознак і прогнозування конверсії користувачів мобільного застосунку», керівник роботи Недашківська Надія Іванівна, професор кафедри ММСА, доктор технічних наук, професор, затверджені наказом по університету від 27.05.2026 р. №1944-с.
2. Термін подання студентом роботи «10» червня 2026 р.
3. Вихідні дані до роботи: гетерогенні логи подій користувачів мобільного застосунку, маркетингові атрибуційні дані, фреймворки для потокової обробки великих масивів даних, алгоритми обробки природної мови, ансамблеві моделі градієнтного бустингу, методи пояснювального штучного інтелекту.
4. Зміст роботи: дослідження предметної області та аналіз існуючих підходів до прогнозування конверсії користувачів методами машинного навчання, математичне та алгоритмічне забезпечення системи прогнозування конверсії, програмна реалізація системи прогнозування конверсії та порівняльний аналіз моделей машинного навчання, функціонально-вартісний аналіз апаратно-програмного засобу та програмного продукту.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): UML-діаграми архітектури та функціональної роботи конвеєра даних (рис.), зведені таблиці метрик якості моделей, матриці помилок, графіки

кривих навчання та калібрування (рис.), візуалізації глобальної та локальної інтерпретації SHAP, презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Функціонально-вартісний аналіз програмного продукту	Рощина Н.В., доц., к.е.н.		

7. Дата видачі завдання 20.02.2026 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи (ДР)	Термін виконання етапів роботи	Примітка
1	Затвердження теми ДР.	20.02.2026– 06.03.2026	Виконано
2	Ознайомлення з ДСТУ 3008:2015.	07.03.2026– 09.04.2026	Виконано
3	Проведення дослідження за заданою темою.	10.04.2026– 14.04.2026	Виконано
4	Формалізація запропонованого методу.	15.04.2026– 24.04.2026	Виконано
5	Підготовка першого розділу.	25.04.2026– 30.04.2026	Виконано
6	Підготовка другого розділу.	01.05.2026– 09.05.2026	Виконано
7	Проведення роботи над практичною частиною ДР.	10.05.2026– 13.05.2026	Виконано
8	Підготовка третього розділу.	14.05.2026– 19.05.2026	Виконано
9	Підготовка четвертого розділу.	20.05.2026– 01.06.2026	Виконано
10	Оформлення ДР та аналіз отриманих результатів.	02.06.2026– 09.06.2026	Виконано

Студентка

Богдана ГАВРИЛЮК

Керівник

Надія НЕДАШКІВСЬКА

РЕФЕРАТ

Дипломна робота: 230 с., 24 рис., 16 табл., 4 дод., 35 дж.

ПРЕДИКТИВНА АНАЛІТИКА, МАШИННЕ НАВЧАННЯ, БІНАРНА КЛАСИФІКАЦІЯ, ГРАДІЄНТНИЙ БУСТИНГ, ПОВЕДІНКОВІ ПАТЕРНИ, ОБРОБКА ВЕЛИКИХ ДАНИХ, ІНТЕРПРЕТОВАНІСТЬ МОДЕЛЕЙ.

Предмет дослідження — методи конструювання ознак, алгоритми обробки великих даних та ансамблеві моделі машинного навчання для задач бінарної класифікації поведінкових даних.

Об'єкт дослідження — процес конверсії користувачів мобільних додатків та їхні поведінкові патерни.

Мета дослідження. — підвищення ефективності залучення та монетизації користувачів мобільного застосунку шляхом розробки системи прогнозування їхньої конверсії у платну підписку після завершення тестового періоду.

Використані моделі — XGBoost, LightGBM, CatBoost.

Актуальність роботи зумовлена потребою проактивного оцінювання якості трафіку в перші години використання додатка, що дозволяє оптимізувати рекламні бюджети в реальному часі, не очікуючи завершення тестового періоду користувачів.

Отриманні результати — побудована модель предиктивної аналітики, що може прогнозувати ймовірність конверсії користувачів мобільного застосунку на ранніх етапах з прийнятною швидкістю та точністю.

В рамках подальшого дослідження пропонується налаштувати автоматичну передачу прогнозних сигналів у маркетингові платформи через Conversions API, що прискорить навчання рекламних ML-алгоритмів і забезпечить мікрооптимізацію таргетингу на аудиторію з найвищою ймовірністю конверсії.

ABSTRACT

Thesis: 230 p., 24 fig., 16 tabl., 4 app., 35 ref.

PREDICTIVE ANALYTICS, MACHINE LEARNING, BINARY CLASSIFICATION, GRADIENT BOOSTING, BEHAVIORAL PATTERNS, BIG DATA PROCESSING, MODEL INTERPRETABILITY.

Research topic — feature engineering methods, big data processing algorithms, and ensemble machine learning models for binary classification tasks involving behavioral data.

Research object — the conversion process of mobile app users and their behavioral patterns.

Research objective — to improve the effectiveness of user acquisition and monetization for a mobile app by developing a system to predict their conversion to a paid subscription after the trial period ends.

Models used — XGBoost, LightGBM, CatBoost.

The relevance of this work stems from the need for proactive assessment of traffic quality during the first hours of app usage, which allows for real-time optimization of advertising budgets without waiting for the end of the user trial period.

Results obtained — a predictive analytics model was built that can predict the probability of mobile app user conversion in the early stages with acceptable speed and accuracy.

As part of further research, we propose configuring the automatic transmission of predictive signals to marketing platforms via the Conversions API, which will accelerate the training of advertising ML algorithms and ensure micro-optimization of targeting toward the audience with the highest probability of conversion.

ЗМІСТ

ВСТУП.....	11
1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО ПРОГНОЗУВАННЯ КОНВЕРСІЇ КОРИСТУВАЧІВ МЕТОДАМИ МАШИННОГО НАВЧАННЯ	12
1.1 Вступ до предметної області та економіка мобільних підписок.....	12
1.2 Концептуалізація поведінки користувачів	15
1.3 Метрики фрустрації користувачів як предиктори відмови від конверсії	17
1.4 Попередня обробка гетерогенних даних.....	19
1.4.1 Обробка пропусків у гетерогенних логах	19
1.4.2 Боротьба з незбалансованістю даних	20
1.5 Зменшення витрат обчислювальних ресурсів при обробці великих масивів логів	22
1.6 Конструювання поведінкових ознак: адаптація методів обробки природної мови та метрика TF-IDF	24
1.7 Моделі машинного навчання для бінарної класифікації поведінкових даних	26
1.7.1 Вимоги до конвеєрності: швидкість та легкість моделей	27
1.7.2 Вимоги до інтерпретованості результатів	27
1.7.3 Градієнтний бустинг	28
1.8 Інтеграція прогнозних моделей у маркетингові екосистеми: Conversions API та розподілення бюджетів	30
1.8.1 Механіка автоматизованого зворотного зв'язку через CAPI	31
1.9 Постановка задачі дослідження	33

	7
1.10 Висновки до розділу 1	36
2 МАТЕМАТИЧНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ПРОГНОЗУВАННЯ КОНВЕРСІЇ	38
2.1 Математична формалізація архітектури обробки надвеликих масивів подій.....	38
2.1.1 Математичне моделювання ефективності доступу до колонкових даних.....	38
2.1.2 Алгоритмічна оптимізація планів запитів на основі спрямованих ациклічних графів.....	41
2.1.3 Моделювання паралельного виконання за алгоритмом Work-Stealing	43
2.2 Математичне моделювання поведінкових патернів та метрик фрустрації	44
2.2.1 Формалізація індексів успішності сценаріїв.....	45
2.2.2 Математична оцінка когнітивного зусилля та фрустрації	45
2.2.3 Інформаційно-теоретична оцінка хаотичності дій.....	47
2.3 Теоретичні засади векторизації та трансформації ознак	47
2.3.1 Статистичне обґрунтування метрики TF-IDF для поведінкових послідовностей	48
2.3.2 Контекстуалізація через n-грами та динамічне зважування	49
2.3.3 Математичні методи стабілізації дисперсії	50
2.4 Алгоритмічне обґрунтування методів градієнтного бустингу	51
2.4.1 Математична постановка задачі адитивного навчання	51
2.4.2 XGBoost: Оптимізація другого порядку Тейлора	52
2.4.3 LightGBM: Градієнтний однобічний семплінг та об'єднання ознак ..	53
2.4.4 CatBoost: Впорядковані цільові статистики та симетричні дерева ..	54
2.4.5 Математичне обґрунтування валідації в часі	55
2.5 Математичне обґрунтування інтерпретованості та калібрування результатів.....	57

	8
2.5.1 Теорія ігрових значень Шеплі.....	58
2.5.2 Математична оптимізація порогу прийняття рішень для маркетингових сигналів.....	59
2.6 Висновки до розділу 2.....	63
3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ПРОГНОЗУВАННЯ КОНВЕРСІЇ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ...	65
3.1 Інструментальні засоби та обчислювальна архітектура.....	65
3.1.1 Мова програмування та обчислювальне ядро	66
3.1.2 Бібліотеки машинного навчання.....	69
3.1.3 Середовище розробки та апаратне забезпечення.....	71
3.2 Розробка конвеєра обробки даних	75
3.2.1 Збір, фільтрація та очистка сирих логів	77
3.2.2 Логіка генерації статичних та часових ознак	82
3.2.3 Програмна реалізація багаторівневої таксономії та метрик фрустрації.....	86
3.2.4 Векторизація поведінкових послідовностей.....	89
3.3 Відбір ознак та формування фінального високорозмірного датасету....	92
3.3.1 Теоретичне та архітектурне обґрунтування парадигми "раннього злиття"	93
3.3.2 Управління оперативною пам'яттю під час конкатенації розріджених та щільних матричних структур	95
3.3.3 Математична трансформація та нормалізація числового простору .	97
3.3.4 Дисперсійний аналіз та алгоритмічна елімінація квазіконстантних ознак.....	101
3.3.5 Симбіоз високорозмірного простору з архітектурою алгоритмів градієнтного бустингу.....	103
3.3.6 Структурно-семантична декомпозиція фінального високорозмірного датасету	104

3.4 Порівняльний аналіз моделей градієнтного бустингу для класифікації конверсії	107
3.4.1 Дизайн експерименту та метрики оцінювання моделі	108
3.4.2 Загальні налаштування архітектури навчання моделей	111
3.4.3 Навчання та налаштування гіперпараметрів моделі XGBoost.....	113
3.4.4 Навчання та налаштування гіперпараметрів моделі LightGBM	116
3.4.5 Навчання та налаштування гіперпараметрів моделі CatBoost.....	120
3.4.6 Порівняльна таблиця результатів експериментів	124
3.5 Інтерпретація моделі та імітація бізнес-інтеграції.....	128
3.5.1 Глобальна та локальна інтерпретація результатів	129
3.5.2 Калібрування порогу відсічення для Conversions API.....	138
3.6 Висновки до розділу 3.....	144
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ АПАРАТНО-ПРОГРАМНОГО ЗАСОБУ ТА ПРОГРАМНОГО ПРОДУКТУ	146
4.1 Постановка задачі проектування	146
4.2 Обґрунтування функцій програмного продукту	148
4.3 Обґрунтування системи параметрів програмного продукту.....	153
4.4 Аналіз експертного оцінювання параметрів.....	157
4.5 Аналіз рівня якості варіантів реалізації функцій	162
4.6 Економічний аналіз варіантів розробки ПП	165
4.7 Вибір кращого варіанту розробки за критерієм техніко-економічного рівня	169
4.8 Висновки до розділу 4.....	171
ВИСНОВКИ	173
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	175
ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО КОДУ	180

	10
ДОДАТОК Б АРХІТЕКТУРА РОЗРОБЛЕНОЇ СИСТЕМИ	214
ДОДАТОК В ФУНКЦІОНАЛЬНА СХЕМА РОБОТИ СИСТЕМИ	215
ДОДАТОК Г ПРЕЗЕНТАЦІЯ.....	216

ВСТУП

За останні роки спостерігається стрімкий розвиток екосистеми мобільних застосунків, що супроводжується фундаментальним переходом індустрії від транзакційної моделі монетизації до моделі регулярних підписок. Зокрема, для утилітарних продуктів критичного значення набуває здатність швидко донести свою практичну цінність до користувача в перші години чи навіть хвилини використання. Однак в умовах жорсткої конкуренції та постійного дорожчання рекламного трафіку бізнес стикається з масовим скасуванням безкоштовних пробних періодів через когнітивне перевантаження, фрустрацію або нерозуміння інтерфейсу користувачами. Традиційні методи ретроспективної аналітики, які спираються на усереднені історичні показники, виявляються занадто повільними та економічно неефективними для динамічного управління маркетинговими бюджетами і якістю залученого трафіку [1].

Предиктивна аналітика та алгоритми машинного навчання сьогодні дозволяють обробляти надвеликі масиви гетерогенних логів і автоматично виявляти приховані закономірності в поведінці користувачів. У цій роботі буде розглянуто процес розробки, інженерного та математичного обґрунтування масштабованої системи прогнозування конверсії користувачів мобільного застосунку в платну підписку. Основна увага буде приділена трансформації сирих дискретних подій у неперервні метрики когнітивного зусилля, векторизації хронологічних послідовностей за допомогою методів обробки природної мови, а також застосуванню алгоритмів градієнтного бустингу. Це слугуватиме фундаментом для створення механізму ранньої ідентифікації платоспроможної аудиторії та автоматизованого покращення продуктивності рекламних кампаній через серверні протоколи інтеграції у режимі реального часу.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО ПРОГНОЗУВАННЯ КОНВЕРСІЇ КОРИСТУВАЧІВ МЕТОДАМИ МАШИННОГО НАВЧАННЯ

1.1 Вступ до предметної області та економіка мобільних підписок

Теперішній ринок мобільних додатків проходить етап суттєвих змін, зміщуючи фокус від транзакційної моделі монетизації, яка зосереджена на разових покупках користувачів, до підписочної з регулярним списанням коштів. Згідно з глобальними галузевими звітами, зокрема звітом State of Mobile 2026, фактичний загальний обсяг ринку покупок у мобільних додатках за результатами 2025 року досягнув 167 мільярдів доларів США, перевершуючи прогнозований показник у розмірі 160 мільярдів. Враховуючи невідомі темпи розвитку індустрії, залучення нових користувачів на продукти стає все більш складним та нерівномірним. Вартість закупки рекламного трафіку невідомо дорожчає, а очікування користувачів від програмних продуктів зростають [1].

Сфера мобільних додатків практикує доволі багато моделей монетизації, проте найбільш популярною є підписочна в комбінації з фріміум. Така модель дозволяє користувачам використовувати обмежений функціонал додатків безкоштовно з преміум доступом за фіксовану суму на певний період. Доступ до розширеного функціоналу надається користувачу для ознайомлення безкоштовно і зазвичай триває 3 або 7 днів, це називається тріалом, після чого починаються регулярні списання коштів за фіксовані проміжки часу, які в основному складають 7 чи 30 днів. Метою такого підходу є зниження бар'єру адаптації користувача на новому для нього продукті шляхом надання йому можливості оцінити цінність додатку.

Проте патерни поведінки користувачів кардинально відрізняються залежно від сфери додатку. Якщо розважальні платформи, як Netflix чи Spotify, орієнтовані на довгострокове споживання контенту та поступове формування звички, то утиліти, додатки у сфері Health & Fitness або вузькоспеціалізовані продукти, наприклад, ботанічні застосунки, мають іншу природу. Користувачі завантажують їх із високою мотивацією для вирішення конкретної проблеми «тут і зараз». Для таких продуктів важливим є швидке досягнення так званого "Aha-моменту" – миті, коли людина відразу розуміє практичну цінність інструменту для свого завдання. Тому основна взаємодія та оцінка корисності відбувається безпосередньо під час онбордингу та в перші години використання [2].

Подібні бізнес-моделі містять у собі суттєвий недолік, який полягає у небажанні користувачів сплачувати за отриманий функціонал, що спричиняє масове скасування підписок на етапі тріалу, внаслідок чого відсоткове співвідношення таких користувачів суттєво переважає, порівняно з майбутніми платниками. Специфіка утилітарних додатків зумовлює те, що користувач може або швидко розв'язати свою разову проблему і піти, або не відчувати миттєвої цінності та відмовитися від продукту. Статистично, найвищий показник скасування підписок припадає на день взяття тріалу, або ж нульовий день використання. Також значна частка користувачів використовує банківську картку, на якій замало коштів, щоб оплатити підписку. Платіжні системи маркетплейсів адаптовані під це і намагаються списати кошти ще протягом певного періоду після завершення безкоштовного періоду, який сягає 30 днів.

Всі вищеописані фактори формують одне із найважливіших завдань для бізнесів, яке полягає в ефективному управлінні конверсії користувачів з безкоштовного періоду в оплату. У ніші утиліт та здоров'я, на відміну від стрімінгових сервісів, рішення про необхідність продукту приймається блискавично, тому управління конверсією та утриманням саме в ці перші години стає питанням економічного виживання бізнесу. Це створює завдання

як технічній частині бізнесу, що розробляє функціонал додатку, який приносить цінність користувачу та перевершуватиме конкурентів, так і маркетинговій, де важливо відразу розуміти якість закупленої аудиторії. З огляду на це, управління конверсією та утриманням у ці перші години стає найважливішим завданням для забезпечення життєздатності бізнесу.

Прогнозні моделі для передбачення конверсії в оплату чи утримання користувачів мають широке використання. Наприклад, Netflix використовує машинне навчання для аналізу тривалих сесій перегляду та побудови систем персоналізованих рекомендацій, що є їхнім головним інструментом стимулювання лояльності. Натомість для застосунків, що вирішують конкретні утилітарні завдання, прогнозна аналітика повинна фокусуватися на іншому: на аналізі ранніх мікро-взаємодій, швидкості активації ключових функцій та виявленні сліпих зон або фрустрації користувача під час найпершого сеансу.

Традиційна аналітика спирається на історичні значення на етапі закупки, тобто в період тріалу для користувача, коли береться середній показник за минуле і вважається, що нові користувачі ймовірно поводитимуться так само, після чого аналізуються отримані фактичні значення. Такий підхід має суттєву перевагу у своїй простоті й відсутності потреби у складних обчислювальних потужностях, проте він є доволі економічно неефективним через усереднення результатів, накопичення похибки та суттєве обмеження регулювання закупки в реальному часі. Також одним із головних недоліків є нездатність такого методу адаптуватися до динамічних змін у поведінці користувачів. Це унеможлиблює оптимізацію маркетингових кампаній в режимі реального часу та може призводити до неефективного витрачання бюджетів на нецільовий трафік.

Враховуючи це, виникає потреба переходу з примітивних методів прогнозування до комплексних з імплементацією методів машинного навчання, здатних миттєво обробляти специфічні ранні поведінкові сигнали цільової аудиторії.

З іншого боку, з розвитком предиктивної аналітики, широкого застосування набули глибокі нейронні мережі. Їхньою головною перевагою є здатність автоматично виявляти складні нелінійні залежності. Однак у контексті мобільної аналітики утилітарних додатків, де дані переважно є табличними, гетерогенними, розрідженими та містять високу частку пропусків, нейромережі демонструють суттєві архітектурні недоліки, адже тоді вони схильні до перенавчання на тренувальних вибірках з екстремальним дисбалансом класів і вимагають значних обчислювальних ресурсів для розгортання у production-середовищі. Крім того, природа нейромереж як "чорної скриньки" робить їх слабо інтерпретованими, що ускладнює розуміння бізнесом того, які саме поведінкові фактори чи елементи UX-дизайну спричиняють конверсію.

Більш прості алгоритми машинного навчання, такі як одиничні дерева рішень, забезпечують високу прозорість та легкість інтерпретації результатів. Незважаючи на це, базові дерева рішень є вкрай чутливими до невеликих змін у вхідних даних, адже мають високу дисперсію, та часто стають упередженими на користь мажоритарного класу при роботі з незбалансованими даними. Для подолання цих недоліків оптимальним рішенням стає застосування ансамблевих методів градієнтного бустингу. Вони ефективно компенсують слабкі сторони одиничних дерев шляхом послідовної мінімізації помилок, забезпечуючи високу швидкість обробки, стійкість до пропусків у логах та точність прогнозування, що є необхідним для конвеєрної аналітики.

1.2 Концептуалізація поведінки користувачів

Для побудови надійних прогнозних моделей необхідно формалізувати об'єкт дослідження, що в даному контексті є поведінка та дії користувача в додатку. Поведінковими даними є хронологічно позначені події, які виконував

користувач протягом періоду безпосередньої активності в додатку. Це дозволяє дослідити тривалість певних подій та порядок їх виконання.

У науковій літературі, присвяченій аналізу поведінкових патернів, загальний контекст взаємодії користувача з додатком традиційно розглядається крізь призму кількох ключових вимірів. Першим із них є активна поведінка, що охоплює свідомі дії, спрямовані на зміну стану системи, такі як налаштування фільтрів, використання пошуку, відправлення повідомлень, а в спеціалізованих додатках — проходження тестів чи додавання платіжних карток. На противагу їй виділяють пасивну поведінку, яка полягає виключно зі споживанні контенту: перегляді інформаційних екранів, читанні умов послуг або ознайомленні з тарифами без активної взаємодії. Третім невід'ємним виміром виступає просторово-часовий та демографічний контекст, що охоплює зовнішні фактори сесії, зокрема час доби, день тижня, характеристики пристрою, країну резиденції та джерело залучення трафіку. Саме комплексне врахування всіх цих активних, пасивних та контекстних показників дозволяє предиктивним алгоритмам ефективно знаходити нелінійні кореляції та прогнозувати наміри щодо покупки.

Окрім самих типів поведінки, важливим виміром для побудови предиктивної моделі є послідовність дій, тобто необхідно враховувати не лише наявність певної події, а й її точне місце у хронологічному ланцюжку користувацького шляху. Наприклад, послідовність дій `onboarding` → `paywall` → `scan` свідчить про один патерн ознайомлення, де користувач спочатку бачить умови та ціну, а потім тестує функціонал, що працює інакше і вказує на іншу ймовірність конверсії, ніж послідовність `scan` → `onboarding` → `paywall`.

У зв'язку із раннім скасуванням підписок та з погодинним зниженням показником повернення на продукт, поведінкові патерни користувачів за перші години активності з преміум доступом є ключовими для прогнозування їх майбутніх дій, що в даному контексті є подальша оплата чи скасування підписки. Саме в цей період вмотивованість користувачів до використання функціоналу є найвищою, з чого можна отримати цінні висновки. Крім

виконання позитивних дій, які добре впливають на користувацький досвід, фундаментальним аспектом прогнозування відтоку клієнтів є виявлення патернів, які сигналізують про негативний користувацький досвід та труднощі у використанні інтерфейсу.

У науковій літературі з поведінкової аналітики та статистики когорта визначається як сукупність індивідів, які об'єднані спільними демографічними чи поведінковими характеристиками або зазнали однакової події, наприклад, реєстрації чи першої покупки протягом визначеного часового інтервалу. Формування таких груп є фундаментальним методом для відстеження змін у поведінці користувачів з плином часу та виявлення специфічних патернів для прийняття рішень. У межах даного дослідження під когортою розуміється жорстко сегментована група користувачів, які одночасно задовольняють наступні критерії: активували безкоштовний пробний період в один і той самий фіксований день, були залучені на продукт з одного конкретного медіа джерела та є резидентами однієї країни. Крім того, для мінімізації апаратного шуму, забезпечення максимальної однорідності поведінкових патернів та структури подій, емпірична база дослідження обмежується виключно логами користувачів операційної системи iOS.

1.3 Метрики фрустрації користувачів як предиктори відмови від конверсії

У сфері взаємодії людини з додатками та дизайну користувацького досвіду зростає розуміння того, що кількісні метрики залученості, такі як тривалість сесії, не завжди корелюють із задоволеністю. Довга сесія може свідчити не про глибоку зацікавленість, а про неможливість користувача знайти потрібну функцію або завершити цільову дію. Відповідно, виникає потреба в ідентифікації та формалізації стану фрустрації користувача.

Фрустрація розглядається як емоційний стан, спричинений когнітивним перевантаженням або системними перешкодами, що заважають досягненню мети. Алгоритми машинного навчання здатні виявляти аномалії у поведінкових послідовностях, що корелюють із фрустрацією, і, як наслідок, з високою ймовірністю видалення додатка [3].

Одним із таких патернів можна розглядати значне відхилення часу перебування на певному екрані від медіанного значення для даної когорти користувачів. Емпіричні дослідження у сфері поведінкової аналітики вказують на пряму кореляцію між часом, проведеним у стані розгубленості, та відмовою від покупки. Особливу увагу варто приділяти конкретним проявам технічної та поведінкової фрустрації, що безпосередньо фіксуються у логах. До таких індикаторів належать серії системних або мережових помилок, наприклад подія `app_error`, неможливість знайти потрібний контент чи функцію — подія `search_failed`, а також переривання цільових користувацьких процесів на півшляху, наприклад, скасування запиту до фахівця-ботаніка, наприклад подія `ask_botanist_cancelled`. Також варто звертати увагу на серії івентів помилок та періодичне повернення користувача на екран управління підпискою, що може сигналізувати про намір скасування підписки.

Інтеграція подібних оцінок та коефіцієнтів психологічного зусилля, які взаємодіють з перешкодами, демонструє свою ефективність у прогнозуванні утримання користувачів шляхом навчання моделей на негативному користувацькому досвіді. У розроблюваній системі прогнозування конверсії розрахунок відносних показників фрустрації замість використання простих лічильників є важливою вимогою до етапу конструювання ознак, що дозволить моделі машинного навчання розрізняти залученість та розгубленість. Проте важливо уникати використання подій, які однозначно призводять до скасування підписки, для уникнення перенавчання моделі.

1.4 Попередня обробка гетерогенних даних

Обробка логів мобільних додатків пов'язана з двома фундаментальними проблемами: неоднорідністю вхідного потоку, що призводить до великої кількості пропущених значень, та незбалансованістю цільового класу. Крім того, робота з мільйонами записів подій створює значне навантаження на обчислювальні ресурси, адже зі зростанням обсягів даних виникає постійний ризик переповнення оперативної пам'яті. Сучасні практики індустрії передбачають використання інструментів з колонковою архітектурою пам'яті, наприклад, на базі формату Apache Arrow, та лінійних обчислень, що дозволяє оптимізаторам запитів обробляти масиви даних потоково без створення надлишкових проміжних копій у пам'яті.

1.4.1 Обробка пропусків у гетерогенних логах

Через використання реальних даних про події, виконані користувачами, відсутність певної дії, наприклад, ненажаття кнопки чи невідвідування екрану, часто створює ілюзію наявності пропущених значень. Однак у подієвій аналітиці відсутність факту події це не класичний пропуск у традиційному розумінні реляційних таблиць, а природна властивість формування даних, що називається розрідженістю.

Застосування академічної теорії механізмів виникнення пропусків, таких як MCAR, MAR, MNAR, та традиційних методів імпутації, наприклад, заповнення статистичним середнім, до показників частоти івентів є концептуально неправильним та неприпустимим. Натомість послідовності поведінкових подій формують розріджені матриці, де переважна більшість значень дорівнює нулю, оскільки окремий користувач фізично взаємодіє лише

з невеликою часткою усього доступного функціоналу додатка протягом однієї сесії.

Інформативна відсутність у такому контексті набуває дещо іншого змісту: сам факт того, що подія не відбулася, тобто нульове значення у розрідженій матриці, вже є сигналом про конкретний поведінковий патерн і слугує важливим джерелом інформації. У випадках, коли пропуски виникають саме в категоріальних метаданих події, наприклад, невідома країна або модель пристрою, слід створювати окремі категорії "unknown". Такий підхід дозволяє деревовидним моделям вловлювати приховані закономірності, виходячи з самої відсутності цих метаданих.

Для забезпечення високої швидкості конвеєра використовується інструмент потокової обробки даних, що дозволяє ефективно агрегувати розріджені матриці без перевантаження оперативної пам'яті. Сучасні алгоритми градієнтного бустингу від самого початку спроектовані для вирішення цих проблем. Зокрема, алгоритми як XGBoost та LightGBM мають вбудовані математичні оптимізації для роботи з розрідженими просторами ознак, а CatBoost здатний автоматично обробляти відсутні значення в категоріальних ознаках під час тренування, використовуючи цільові статистики. Це повністю позбавляє необхідності створювати складні та ресурсомісткі вузли препроцесингу перед подачею даних у модель [4].

1.4.2 Боротьба з незбалансованістю даних

Прогнозування конверсії у першу оплату після безкоштовного періоду є класичною задачею з незбалансованими даними. Відсоток користувачів, які успішно здійснюють першу оплату, є міноритарним класом, тобто у разі меншим за тих, хто превентивно скасовує підписку, що є мажоритарним класом. Навчання базових моделей на таких даних часто призводить до того,

що алгоритм просто починає ігнорувати міноритарний клас, покращуючи загальну точність шляхом передбачення на користь мажоритарного класу, не враховуючи при цьому реальні поведінкові патерни [5].

Для врегулювання незбалансованості класів у задачі прогнозування конверсії використовуються виключно безпечні для високорозмірних даних методи: алгоритмічні адаптації та калібрування порогу прийняття рішення. Важливо зазначити, що класичні методи ресемплінгу, такі як штучна генерація даних SMOTE або ADASYN, є непридатними для роботи з розрідженими TF-IDF векторами та високорозмірними категоріальними даними. Оскільки метод SMOTE створює "синтетичних" сусідів шляхом лінійної інтерполяції, його застосування до текстових чи поведінкових векторів призводить до генерації неіснуючих, аномальних профілів. Це повністю руйнує логіку простору ознак. Тому єдиними правильними підходами залишаються алгоритмічні адаптації та калібрування порогу прийняття рішень [6].

Підхід алгоритмічний адаптацій означає зміну функції втрат самої моделі шляхом застосування зважування класів, де помилкова класифікація користувача, що міг би конвертуватися, "коштує" алгоритму значно дорожче. Методи градієнтного бустингу дозволяють нативно передавати ці ваги, наприклад, через параметр `scale_pos_weight`, і є особливо стійкими до незбалансованості. Крім того, в процесі ітеративної побудови дерев вони природно збільшують вагу для тих зразків, на яких попередні дерева помилилися.

Калібрування порогу прийняття рішення означає динамічний зсув порогу відсічення ймовірності, який за замовчуванням складає 0.5, не змінюючи при цьому алгоритми чи параметри самої моделі. Емпіричні дослідження показують, що налаштування порогу для покращення цільових метрик є одним із найбільш послідовних та ефективних рішень у задачах з незбалансованою конверсією [7].

1.5 Зменшення витрат обчислювальних ресурсів при обробці великих масивів логів

Стандартною бібліотекою для числових обчислень на мові програмування Python є Pandas. Ця бібліотека базується на масивах бібліотеки NumPy. Історично, вони створювались для числових обчислень для відносно малих об'ємів даних, що означає значне використання оперативної пам'яті при обробці текстових та категоріальних типів даних через негайне виконання кожної операції у конвеєрі, генеруючи проміжні копії даних у пам'яті. Враховуючи апріорну структуру даних, яка в основному складається з категоріальних та текстових даних, використання бібліотеки Pandas як основного інструменту обробки даних є недоречним [8].

Альтернативним інструментом може слугувати бібліотека Polars, яка за своїм функціоналом подібна до Pandas, але розрахована на роботу з великими об'ємами даних. Вона використовує колонковий формат пам'яті, що забезпечує локальність кешу та дозволяє використовувати векторизацію на рівні процесора за принципом single instruction for multiple data, або SIMD.

Обробка даних у Pandas упирається в обмеження Global Interpreter Lock мови Python, що змушує всі операції виконуватися в одному потоці, це робить неможливим утилізацію сучасних багатоядерних процесорів. На противагу Polars, завдяки ядру на Rust, за замовчуванням є багатопотоковим, тому складні операції, такі як JOIN десятків мільйонів рядків з демографічними таблицями або операції GROUP BY для агрегацій, розбиваються на фрагменти і виконуються паралельно на всіх доступних ядрах CPU.

Найбільшою перевагою Polars для складних ETL-пайплайнів є наявність режиму лінивих обчислень та вбудованого оптимізатора запитів. Тобто замість покрокового виконання коду, Polars приймає весь ланцюг операцій, будує логічний план запиту і застосовує евристичні оптимізації до початку виконання. Одним із основних механізмів оптимізації є проштовхування

предикатів, коли фільтри переміщуються найближче до джерела читання даних наскільки це можливо. Замість того, щоб зчитувати весь файл у пам'ять, а потім фільтрувати його, Polars зчитує лише ті рядки, що відповідають наперед заданій умові. Цей механізм призводить до прошовхування проєкцій, коли з джерела даних отримуються лише ті колонки, які безпосередньо необхідні для агрегації чи моделювання, ігноруючи надлишкові метадані [9].

У таблиці 1.1 підсумовано архітектурні відмінності інструментів для побудови конвеєра обробки даних у межах даного дослідження [10].

Таблиця 1.1 — Порівняльний аналіз інструментів обробки даних Pandas та Polars

Характеристика	Pandas	Polars
Базова мова реалізації	C / Cython / Python	Rust
Формат даних у пам'яті	NumPy	Нативно Apache Arrow, колонковий формат
Модель виконання	Покрокова матеріалізація	Лінива з глобальним оптимізатором запитів
Паралелізм	Однопотоківий	Багатопотоківий
Використання пам'яті	Високе (в 5-10 разів більше від розміру даних на диску)	Оптимізоване (в 2-4 рази більше від розміру даних)
Оптимізація запитів	Відсутня на рівні конвеєра	Predicate pushdown, projection pushdown
Енергоефективність	Низька, високе споживання через час виконання	Висока, зниження споживання до 8 разів

Таким чином, для розв'язання поставленої задачі прогнозування конверсії, яка вимагає обробки великомасштабних масивів розрізнених даних, проєктування архітектури на базі фреймворку Polars є необхідною умовою забезпечення стійкості до браку оперативної пам'яті та досягнення прийнятної швидкості конструювання ознак.

1.6 Конструювання поведінкових ознак: адаптація методів обробки природної мови та метрика TF-IDF

Одним із найважливіших етапів побудови прогнозної моделі машинного навчання є трансформація неструктурованих часових послідовностей логів додатків у структуровані формати, тобто числові вектори, що можуть бути оброблені алгоритмами машинного навчання, на відміну від текстових змінних. Застосування простих методів агрегації, таких як базовий підрахунок частоти подій або бінарне кодування наявності дії, призводить до значної втрати неявної інформації. Базові частотні моделі не здатні розрізняти тривіальні дії, які здійснюють абсолютно всі користувачі, та високоспецифічні, рідкісні дії, які є дуже інформативними для передбачення конверсії.

Для вирішення цієї проблеми у сучасних дослідженнях з аналізу поведінкових послідовностей успішно застосовуються методи обробки природної мови (Natural Language Processing, NLP), де дії користувачів моделюються за аналогією до слів у тексті. Одним із найефективніших статистичних методів вилучення ознак у цьому контексті є метрика TF-IDF (Term Frequency - Inverse Document Frequency). Послідовність подій розглядається як "текст", а окремі івенти — як "слова".

Алгоритм TF-IDF дозволяє оцінити, наскільки важливою є конкретна дія або послідовність дій для профілю певного користувача порівняно з поведінкою всієї глобальної вибірки. У контексті аналізу мобільних додатків компоненти класичної формули адаптуються під специфіку поведінкових логів. Так, терміном (t) виступає окрема дискретна подія, виконана людиною в інтерфейсі. Цей підхід може розширюватися до використання n-грам - послідовностей із кількох пов'язаних подій підряд, що допомагає фіксувати часовий контекст взаємодії. Зокрема, біграма `app_launch` → `plant_scan` передає значно більше інформації про цілеспрямованість користувача, ніж ці ж події,

розглянуті ізольовано. Своєю чергою, документом (d) вважається вся хронологічна послідовність подій, згенерована одним унікальним користувачем протягом визначеного вікна спостереження, а корпусом (D) — повна множина таких послідовностей-документів від усіх користувачів, що формують тренувальну вибірку.

Математична модель TF-IDF розраховується як добуток двох статистичних величин. Перша величина — це частота терміну TF, яка вимірює локальну значимість події для окремого користувача. Вона нормалізується на загальну кількість подій у сесії користувача, щоб уникнути зміщення на користь користувачів з аномально довгими сесіями. Друга величина — це обернена частота документа IDF. Вона вимірює глобальну інформативність події в межах усього корпусу D, логарифмічно знижуючи вагу тих подій, що зустрічаються у профілях більшості користувачів, і підвищуючи вагу унікальних, рідкісних сигналів.

Підсумкова вага ознаки для моделі машинного навчання розраховується як добуток двох компонентів. Детальніше про математичну модель розрахунку міститься в підрозділі 2.3.

Застосування алгоритму TF-IDF для векторного представлення поведінкових сесій суттєво перевершує ефективність звичайних агрегаційних підходів. Дослідження показують, що комбінація специфічного конструювання ознак, наприклад, суміщення n-грам та зважування TF-IDF для посилення дискримінативних термінів, з класичними методами машинного навчання дозволяє досягти надзвичайно високої точності у задачах бінарної класифікації послідовностей. Модель автоматично виділяє з сотень можливих типів івентів ті ключові індикатори, які статистично пов'язані з наміром оформити підписку або, навпаки, з високим ризиком відтоку.

Отже, використання методів NLP для просторової векторизації TF-IDF у комбінації із семантичним тегуванням подій, розрахунком індексів фрустрації та побудовою системи нормалізації на рівні когорт, формує

надійний, високорозмірний простір ознак, який є вхідним масивом даних для моделей бінарної класифікації [11].

1.7 Моделі машинного навчання для бінарної класифікації поведінкових даних

Сформований простір ознак є базою для розв'язання центрального завдання дослідження — побудови алгоритму бінарної класифікації. Метою моделювання є визначення функції $f(X) \rightarrow P$, яка відображає вектор X , що містить агреговані демографічні, статичні маркетингові атрибути та розріджені вектори TF-IDF, у ймовірність P настання цільової події. Цільовою подією вважатимемо бінарну змінну `is_converted`, що позначатиме факт успішної оплати підписки після тріалу.

Специфіка вхідних даних мобільної аналітики створює низку математичних викликів для алгоритмів машинного навчання:

- 1) високий ступінь розрідженості — через використання методології n -грам та TF-IDF більшість значень у векторі користувача дорівнюватимуть нулю, оскільки один користувач фізично не може згенерувати всі існуючі варіанти послідовностей подій;
- 2) гетерогенність даних, адже модель має одночасно обробляти неперервні числові змінні, категоріальні змінні з великою кількістю унікальних значень та текстові подібні вектори;
- 3) незбалансованість класів, через кількість користувачів, які здійснили конверсію (меншість, клас 1), на порядки менша за тих, хто відмовився від підписки (більшість, клас 0).

Емпіричні дослідження, присвячені прогнозуванню поведінки споживачів та прецизійному маркетингу (*precision marketing*), порівнювали ефективність різних алгоритмів класифікації: методів опорних векторів,

штучних нейронних мереж зворотного поширення та методів на основі дерев рішень, зокрема градієнтного бустингу. Результати цих досліджень одноставно свідчать, що хоча глибинне навчання має переваги при роботі з однорідними даними, такими як зображення, аудіо, безперервний текст, для гетерогенних табличних наборів з великою кількістю категоріальних ознак та розріджених даних найкращу результативність демонструють моделі градієнтного бустингу.

1.7.1 Вимоги до конвеєрності: швидкість та легкість моделей

Специфіка впровадження моделі в реальний бізнес-конвеєр створює жорсткі обмеження на архітектуру, оскільки прогноз має генеруватися в перші години активності користувача, важливою є швидкість попередньої обробки сирих даних та низька затримка при отриманні прогнозу.

З огляду на це, використання надскладних глибоких нейронних мереж є недоцільним. Нейронні мережі потребують значних обчислювальних ресурсів для інференсу, а також, при роботі з табличними, розрідженими та гетерогенними даними складні нейромережі схильні до швидкого перенавчання на тренувальних вибірках. Натомість, легкі та швидкі моделі на базі дерев рішень забезпечують необхідний баланс між швидкістю роботи у production-конвеєрі та високою точністю.

1.7.2 Вимоги до інтерпретованості результатів

Окрім проблеми перенавчання та високих вимог до обчислювальних ресурсів, фундаментальним недоліком використання глибоких нейронних

мереж у завданнях бізнес-аналітики є їхня архітектурна непрозорість, відома як проблема «чорної скриньки». У задачах прогнозування конверсії для бізнесу важливо не лише отримати точну кінцеву ймовірність дії користувача, але й чітко розуміти, які саме фактори спонукали його до цього рішення. Складні нейронні мережі приховують нелінійні взаємозв'язки між вхідними поведінковими векторами та прогнозом, що робить практично неможливим прозорий аналіз важливості окремих ознак.

Нераціональність використання таких закритих систем також зумовлена складністю інтерпретації метрик якості в контексті реальних продуктових процесів. Для прийняття рішень маркетологам та продакт-менеджерам необхідне розуміння того, як саме конкретна дія, наприклад, серія помилок або відвідування певного екрану, впливає на готовність оформити підписку. Без пояснювального штучного інтелекту неможливо валідувати, чи модель виявила дійсно значущі поведінкові патерни, чи просто запам'ятала статистичний шум вхідних даних. Це унеможливорює створення ефективних та персоналізованих стратегій утримання користувачів.

На противагу, легкі ансамблеві моделі на базі дерев рішень, зокрема алгоритми градієнтного бустингу, дозволяють застосовувати математичні методи інтерпретації. Це забезпечує як глобальне розуміння впливу кожної характеристики на модель загалом, так і локальне пояснення прогнозу для кожного окремого користувача. Така прозорість дозволяє бізнесу точно ідентифікувати тригери конверсії та фрустрації, що надає алгоритму машинного навчання змісту.

1.7.3 Градієнтний бустинг

У задачах прогнозування схильності до конверсії базові дерева рішень демонструють низку критичних недоліків, що унеможливорює їх використання

як самостійної моделі. Перш за все, одиничні дерева намагаються ідеально підлаштуватися під тренувальні дані, утворюючи глибокі та складні структури, які погано генералізують нові дані. Невеликі зміни у вхідних даних, наприклад зміна поведінки через оновлення інтерфейсу додатка, можуть призвести до кардинальної зміни структури дерева і, як наслідок, нестабільних прогнозів. Також дерева рішень часто схильються до мажоритарного класу під час вибору вузлів розгалуження, що призводить до упереджених прогнозів для міноритарного класу.

Ці обмеження успішно долаються за допомогою методів, де об'єднання багатьох слабких дерев радикально знижує дисперсію та підвищує стійкість до перенавчання. Алгоритм градієнтного бустингу будує прогнозну модель у вигляді ансамблю дерев рішень. На відміну від методу випадкового лісу, де дерева будуються незалежно, у бустингу кожне наступне дерево тренується для мінімізації залишків помилок попередніх дерев, використовуючи метод градієнтного спуску в просторі функцій. Серед сучасних реалізацій цього підходу виділяють XGBoost (Extreme Gradient Boosting) та CatBoost (Categorical Boosting) [12].

XGBoost впроваджує системні оптимізації, як використання інформації про другу похідну (Гессіан) функції втрат для більш точної та швидкої збіжності, а також застосування вбудованої L1 і L2 регуляризації, що запобігає перенавчанню. Крім того, XGBoost містить оптимізовані структури даних для обробки розріджених матриць, що робить його ідеальним для роботи з TF-IDF векторами [13].

CatBoost має фундаментальну перевагу в задачах мобільної аналітики завдяки вбудованому механізму обробки категоріальних змінних. Замість традиційного підходу One-Hot Encoding, який суттєво розширює розмірність матриці при великій кількості унікальних значень категорій, CatBoost використовує алгоритм Ordered Target Statistics, який обчислює очікуване значення цільової змінної для кожної категорії динамічно, спираючись лише

на історичні дані, розташовані вище у штучно відсортованому масиві, тим самим ефективно запобігаючи витоку даних [14].

LightGBM виділяється архітектурою побудови дерев, адже він використовує стратегію росту по листках замість поглиблення по рівнях. Завдяки цьому та використанню гістограмних алгоритмів, LightGBM забезпечує екстремально швидке навчання та інференс, споживаючи менше оперативної пам'яті. У задачах класифікації величезних масивів поведінкових даних LightGBM часто перевершує XGBoost за швидкістю, зберігаючи конкурентну або вищу точність [15].

Відповідно до поставленого завдання дослідження, навчання фінальної моделі здійснюватиметься з використанням алгоритмів градієнтного бустингу, які найкраще відповідають профілю згенерованого простору поведінкових ознак та забезпечують високу предиктивну стійкість [4].

1.8 Інтеграція прогнозних моделей у маркетингові екосистеми: Conversions API та розподілення бюджетів

Практична цінність подібних моделей зокрема полягає у створенні механізму автоматизованої взаємодії з рекламними кабінетами шляхом маршрутизації персоналізованих прогнозних подій, серверних сигналів, для кожного окремого користувача. Передача даних про високу ймовірність конверсії безпосередньо в маркетингові платформи, наприклад, через Conversions API (CAPI), дозволяє ML-алгоритмам рекламних мереж навчатися в рази швидше, що перетворює модель машинного навчання на інструмент покращення таргетингу, який алгоритмічно спрямовує закупівлю на аудиторію з найвищим потенціалом до покупки ще до того, як завершиться їхній тріал.

Більшість підходів оцінки ефективності рекламних кампаній базуються на ретроспективних показниках, таких як вартість залучення одного користувача чи вартість одного встановлення додатку, а оскільки факт монетизації відкладений у часі на тривалість безкоштовного пробного періоду, виникає значний розрив між інвестицією та оцінкою її повернення. Через це однією із важливих навичок маркетолога є професійна інтуїція. Також спеціалісти, в основному, вимушені керуватись статичними правилами розподілу бюджету, що часто може призводити до неефективного використання бюджетів на неякісний трафік.

Застосування предиктивних моделей машинного навчання радикально змінює цю парадигму. Наявність алгоритму, який в режимі реального часу аналізує поведінку користувача у перші години після встановлення додатка та генерує прогноз конверсії, дозволяє здійснювати проактивний розподіл бюджетів. Оцінка якості рекламного каналу, конкретної кампанії чи навіть візуального креативу відбувається не за кількістю залучених користувачів, а за їхньою сумарною прогнозованою ймовірністю конверсії. Алгоритми здатні автоматично перерозподіляти фінансування з кампаній, що постачають користувачів із низьким прогнозним скорингом, на ті кампанії, які генерують якісний цільовий трафік, ще до фактичного завершення їхнього тестового періоду [16].

1.8.1 Механіка автоматизованого зворотного зв'язку через CAPI

Індустрія цифрового маркетингу раніше базувалася на відстеженні поведінки користувачів через фрагменти коду на веб-сайтах або SDK у мобільних додатках, які фіксували дії пристрою на основі файлів cookie або рекламних ідентифікаторів. Проте тепер існують суворі політики конфіденційності користувачів, які все більше набирають популярності через

програми для блокування реклами та створені фреймворки від розробників, таких як Apple, через що точність клієнтського трекінгу погіршується, спричиняючи втрату сигналів поліпшення.

Пристосовуючись до змін, зростає популярність на використання міжсерверних протоколів передачі даних, флагманом яких є API, яке дозволяє рекламодавцям встановлювати пряме захищене з'єднання між власними серверами та серверами рекламних платформ, передаючи інформацію про дії користувачів поза межами обмежень клієнтського браузера чи ОС. Таким протоколом користуються найбільші рекламні платформи як Facebook Ads та Google Ads, якими користується більшість рекламодавців

У контексті даного завдання, архітектура інтеграції моделей машинного навчання з API може бути реалізована за наступним алгоритмом:

- 1) користувач взаємодіє з мобільним додатком, дані про виконані події потоково агрегуються внутрішнім конвеєром;
- 2) після завершення початкового вікна спостереження векторизовані ознаки TF-IDF передаються до навченої моделі;
- 3) модель розраховує ймовірність $P(\text{is_converted})$ і, якщо ймовірність перевищує заздалегідь визначений статистичний поріг, система ідентифікує користувача як більш цінного для продукту, через високу ймовірність подальшої покупки;
- 4) замість того, щоб чекати реальної покупки, бекенд негайно генерує синтетичну подію і відправляє цей сигнал разом із хешованими ідентифікаторами користувача через API до рекламної мережі.

Перевагами такого підходу є те, що передача високоякісних прогнозованих сигналів кардинально прискорює навчання алгоритмів машинного навчання на боці самої рекламної платформи. Рекламні нейромережі отримують інформацію про цільову аудиторію значно швидше, що дозволяє їм оперативно коригувати алгоритми аукціонів та знаходити нових користувачів зі схожими патернами поведінки. У результаті, власна ML-

модель додатка перетворюється на активний інструмент алгоритмічного управління закупівлею трафіку, максимізуючи повернення маркетингових інвестицій за рахунок високоточного раннього виявлення платоспроможної аудиторії [16].

1.9 Постановка задачі дослідження

На основі аналізу предметної області, дослідження ринкових умов та огляду актуальних технологічних інструментів, у даному розділі формалізовано параметри дипломної роботи, визначено її об'єкт, предмет, мету та ключові практичні завдання відповідно до встановлених вимог.

Об'єктом дослідження є процес конверсії користувачів мобільного додатку з розпізнавання та догляду за рослинами, розробленого для платформи iOS, а також специфічні поведінкові патерни, що формуються під час їхньої взаємодії з графічним інтерфейсом продукту.

Предметом дослідження є сукупність методів конструювання ознак, алгоритми потокової обробки великих даних та моделі машинного навчання, застосовані для вирішення задач бінарної класифікації у контексті аналізу часових поведінкових даних.

Мета дослідження полягає у підвищенні загальної ефективності процесів залучення та монетизації користувачів мобільного застосунку шляхом розробки комплексної інтелектуальної системи, яка призначена для прогнозування ймовірності конверсії користувачів у платну підписку після завершення безкоштовного тестового періоду. Основою для формування прогнозу слугуватиме комбінований аналіз статичних атрибутів користувача та деталізованої історії його виконаних подій протягом перших годин активного використання додатка.

Для досягнення зазначеної мети необхідно формалізувати математичну проблему як задачу бінарної класифікації. Метою алгоритму є передбачити ймовірність настання цільової події `is_converted`, що відповідає факту успішної оплати підписки, на базі масиву даних, зібраного в обмежене часове вікно спостереження. Зважаючи на те, що вхідний набір даних характеризується великим обсягом, походить із розрізнених гетерогенних джерел та містить високий ступінь просторової розрідженості, необхідно розробити масштабований програмний комплекс.

Відповідно до сформульованих умов, дослідження передбачає виконання таких основних завдань:

- 1) розробка високонавантаженого потокового конвеєра на базі фреймворку Polars, що забезпечить стійку інтеграцію різнорідних джерел логів та захистить систему від перевантаження оперативної пам'яті;
- 2) реалізація механізмів партиціювання та ефективного батчингу за когортами для зменшення витрат обчислювальних ресурсів процесора;
- 3) здійснення семантичного тегування сирих подій інтерфейсу шляхом розробки багаторівневої ієрархічної таксономії;
- 4) мінімізація використання абсолютних лічильників дій на користь генерації відносних метрик, зокрема розрахунку показників успішності та синтетичних індексів фрустрації користувачів;
- 5) застосування та адаптація методів обробки природної мови, а саме алгоритму TF-IDF, для просторової векторизації поведінкових послідовностей, що дозволить оцінити статистичну значущість унікальних подій, біграм та поведінкових категорій у визначених часових вікнах;
- 6) розробка алгоритму агрегації та розрахунку середніх статистичних показників по когортах, що забезпечить коректну нормалізацію індивідуальної поведінки кожного окремого користувача відносно типового патерну його специфічної групи;

- 7) проєктування, навчання та налаштування гіперпараметрів фінальної моделі градієнтного бустингу з використанням згенерованого гетерогенного простору ознак.
- 8) оцінка моделі для валідації якості бінарної класифікації;
- 9) забезпечення інтерпретованості результатів моделювання за допомогою математичного фреймворку SHAP для глобального аналізу поведінки когорт та локального пояснення індивідуальних предикцій;
- 10) обґрунтування та калібрування оптимального асиметричного порогу прийняття рішень для формування високоочищеного потоку синтетичних конверсій та їх подальшої передачі через Conversions API.

Практична цінність дослідження полягає у тому, що розробка системи дозволить компанії здійснювати проактивну оцінку якості залученого трафіку в перші години після початку використання преміум версії додатка, ліквідувавши необхідність очікування завершення тестового періоду. Зміщення фокусу з ретроспективної фіксації транзакцій на раннє прогнозування дозволить інтегрувати отриману модель з маркетинговими каналами для динамічного розподілу бюджетів.

Робота несе значну технологічну інновацію через створення автоматизованого механізму зворотного зв'язку з рекламними кабінетами. Маршрутизація персоналізованих синтетичних прогнозних подій через протоколи серверної інтеграції СAPI безпосередньо у платформі дозволить алгоритмам закупівлі реклами прискорено навчатися та спрямовувати таргетинг виключно на аудиторію з найвищим конверсійним потенціалом. Це перетворить розроблену ML-модель на центральний інструмент поліпшення економіки продукту, що функціонуватиме в режимі реального часу.

1.10 Висновки до розділу 1

У першому розділі було здійснено дослідження предметної області, присвяченої проблематиці автоматизованого прогнозування конверсії користувачів у мобільних додатках, які функціонуватимуть за моделлю «freemium» та використовують безкоштовні тестові періоди. Визначено, що в умовах зростання конкуренції та вартості залучення клієнтів класичні ретроспективні аналітичні підходи не забезпечують достатньої швидкості реакції для ефективного розподілу маркетингових бюджетів. Натомість інтеграція методів машинного навчання для предиктивної аналітики на етапі ранньої взаємодії з продуктом створює конкурентну перевагу, дозволяючи оцінювати якість трафіку та мінімізувати частку нецільових користувачів в режимі реального часу.

Проведено ґрунтовний аналіз поведінкових патернів, який продемонстрував необхідність виходу за межі стандартних метрик залученості. Виявлено, що дослідження негативного досвіду, так званої фрустрації користувача, є важливим етапом. Формалізація індексів фрустрації і включення їх до загального вектору ознак дозволяє моделі чітко розрізняти ситуації природної залученості та когнітивного перевантаження, що прямо корелює із ймовірністю скасування підписки.

Огляд архітектурних інструментів для обробки великих даних довів, що обробка мільйонів логів подій створює екстремальне навантаження на систему. Доведено доцільність переходу від традиційної бібліотеки Pandas до високоефективної архітектури Polars. Завдяки реалізації на Rust, використанню колонкового формату Apache Arrow, лінівими обчисленням з оптимізацією запитів та нативній багатопотоковості, Polars гарантує захист від помилок оперативної пам'яті, знижує її використання в рази та прискорює ETL-операції, одночасно зменшуючи енергоспоживання системи. Крім того, для вирішення задачі конструювання ознак з неструктурованих

послідовностей обґрунтовано застосування адаптованої NLP-метрики TF-IDF, яка ефективно фільтрує статистичний шум, залишаючи для моделі машинного навчання найбільш дискримінативні, унікальні сигнали користувацької поведінки.

Досліджено теоретичну базу алгоритмів бінарної класифікації. Встановлено, що для роботи з високорозмірними, розрідженими та гетерогенними даними оптимальним вибором є моделі градієнтного бустингу над деревами рішень. Ці алгоритми не лише гарантують максимальні показники F1-Score та площі під кривою ROC AUC, але й дозволяють запобігти витoku цільової змінної при роботі з категоріальними атрибутами, забезпечуючи інтерпретованість результатів для прийняття бізнес-рішень.

Розкрито практичний механізм інтеграції моделі в екосистему управління рекламними бюджетами. Показано, як впровадження протоколу CAPi створює захищений контур для передачі синтетичних предиктивних подій безпосередньо рекламним платформам, що дозволяє прискорити навчання алгоритмів таргетингу самих платформ і автоматизувати пошук найбільш цінної аудиторії. Синтезуючи всі наведені теоретичні та емпіричні дослідження, було структуровано та формалізовано постановку задачі дипломної роботи з чітким визначенням мети, предмета, об'єкта, етапів проєктування та очікуваної практичної цінності. Запропонований системний підхід формує міцне підґрунтя для наступного етапу програмної реалізації архітектури, математичного моделювання та проведення обчислювальних експериментів.

2 МАТЕМАТИЧНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ПРОГНОЗУВАННЯ КОНВЕРСІЇ

2.1 Математична формалізація архітектури обробки надвеликих масивів подій

Розробка надійної, високопродуктивної та масштабованої системи предиктивної поведінкової аналітики вимагає застосування суворого математичного та алгоритмічного обґрунтування на кожному етапі трансформації вхідних даних. Опрацювання гетерогенних хронологічних послідовностей, що налічують десятки мільйонів користувацьких подій, генерує екстремальне навантаження на підсистему оперативної пам'яті та процесорний час. З огляду на це, фундаментальним аспектом побудови обчислювального конвеєра є перехід від традиційних рядкових реляційних парадигм до колонкових обчислень з використанням структур формату Apache Arrow. Крім того, важливим є застосування лінійних оптимізацій через математичний апарат спрямованих ациклічних графів та імплементація алгоритмів динамічного розподілу навантаження багатопотокового середовища за принципом Work-Stealing.

2.1.1 Математичне моделювання ефективності доступу до колонкових даних

Класичні реляційні бази даних та інструменти первинного аналізу даних базуються на рядковій архітектурі зберігання інформації. У цій парадигмі всі атрибути одного об'єкта розміщуються у фізичній оперативній пам'яті або на

жорсткому диску суміжно та послідовно. Математично вартість виконання аналітичного запиту, який передбачає агрегацію або фільтрацію лише за кількома цільовими стовпцями, у рядковій архітектурі моделюється як лінійна функція від загального обсягу всіх даних.

Позначимо загальну кількість записів у наборі даних як N . Нехай кожен запис містить M атрибутів, розміри яких у байтах дорівнюють $s_1, s_2, \dots, s_M$. Тоді загальний розмір одного рядка S_{total} визначається як сума розмірів усіх його атрибутів:

$$S_{total} = \sum_{j=1}^M s_j.$$

Сумарна вартість операцій введення-виведення (I/O) для рядкової бази даних під час виконання аналітичного запиту, який потребує зчитування k цільових атрибутів, де $k \ll M$, завжди дорівнює повній вартості сканування всієї таблиці:

$$C_{row} = N \cdot S_{total} = N \cdot \sum_{j=1}^M s_j. \quad (2.1)$$

Ця формула демонструє суттєву обчислювальну неефективність. Система змушена переміщувати з диска або оперативної пам'яті до кешу процесора весь масив байтів, навіть якщо алгоритм потребує лише незначну частку цих атрибутів для виконання операції групування. Натомість формат Apache Arrow використовує колонкову архітектуру пам'яті, де значення одного атрибута для всіх N записів зберігаються як один безперервний, щільний масив байтів.

Вартість операції введення-виведення у колонковій архітектурі пропорційна виключно розмірам тих колонок, які безпосередньо беруть участь у запиті:

$$C_{col} = N \cdot \sum_{j \in K} s_j, \quad (2.2)$$

де K — множина індексів атрибутів, які запитуються. Оскільки виконується

нерівність $N \cdot \sum_{j \in K} s_j \ll \sum_{j=1}^M s_j$, досягається радикальне зниження навантаження на шину пам'яті та дискову підсистему.

Окрім прямої мінімізації накладних витрат I/O , колонкова модель даних забезпечує ідеальну просторову локальність кешу процесора рівнів L1, L2 та L3. Оскільки дані одного типу, наприклад, масив 64-бітних цілих чисел, розташовані послідовно у пам'яті, центральний процесор здатний застосовувати векторизацію обчислень за апаратним принципом SIMD — Single Instruction, Multiple Data. Принцип SIMD дозволяє арифметико-логічному пристрою процесора завантажувати цілий блок однотипних даних у розширені векторні реєстри і виконувати математичну або логічну операцію над усіма елементами блоку одночасно, витрачаючи на це лише один процесорний такт. У рядковому форматі гетерогенне чергування типів даних руйнує локальність та унеможлиблює використання інструкцій SIMD, змушуючи процесор простоювати в очікуванні даних через часті промахи кешу. Порівняльний математичний та обчислювальний аналіз архітектур пам'яті наведений у таблиці 2.1 [10].

Таблиця 2.1 — Порівняльний математичний та обчислювальний аналіз архітектур пам'яті

Характеристика	Рядкова архітектура	Колонкова архітектура
Фізичне розміщення	Суміжне розміщення атрибутів одного запису	Суміжне розміщення одного атрибута для всіх записів
Вартість запиту	$C_{row} = N \cdot \sum_{j=1}^M s_j$ (за формулою (2.1))	$C_{col} = N \cdot \sum_{j \in K} s_j$ (за формулою (2.2))

Продовження таблиці 2.1

Характеристика	Рядкова архітектура	Колонкова архітектура
Локальність кешу	Низька, забруднення кешу непотрібними атрибутами	Висока, кешуються лише цільові дані
Оптимізація процесора	SISD (Single Instruction, Single Data)	SIMD (Single Instruction, Multiple Data)
Рівень стиснення	Низький, гетерогенні типи даних у блоці	Високий, гомогенні масиви значень

2.1.2 Алгоритмічна оптимізація планів запитів на основі спрямованих ациклічних графів

Обробка надвеликих обсягів інформації вимагає відмови від жадібного виконання кожної програмної інструкції на користь парадигми лінійних обчислень, яка реалізована у сучасних аналітичних рушіях, зокрема Polars. Процес обробки та трансформації даних розглядається як процедура побудови спрямованого (або орієнтованого) ациклічного графа — Directed Acyclic Graph, DAG. У цьому графі вершини представляють формальні оператори реляційної алгебри, а спрямовані ребра (орієнтовані ребра або дуги) відображають логічний потік даних між цими операторами.

Аж до моменту виклику методу-термінатора, глобальний оптимізатор запитів аналізує всю топологію графа обчислень і застосовує евристичні та логічні правила оптимізації, здійснюючи семантично еквівалентні трансформації, які спрямовані на радикальну мінімізацію кількості кортежів

на ранніх етапах виконання. Основними математичними операціями реляційної алгебри, які підлягають оптимізації, є фільтрація, яка позначається як σ , та проєкція, яка позначається як π .

В термінах реляційної алгебри операція селекції записується як $\sigma_c(R)$, де R — відношення, а c — логічний предикат, тобто умова фільтрації. Операція проєкції записується як $\pi_A(R)$, де A — підмножина необхідних атрибутів. Фундаментальним законом оптимізації, який застосовується для переписування графа DAG, є комутативність операцій селекції та проєкції, механізми Predicate Pushdown та Projection Pruning відповідно:

$$\pi_A(\sigma_c(R)) \equiv \sigma_c(\pi_A(R)).$$

Ця тотожність є справедливою за умови, що всі атрибути, які фігурують у логічному предикаті c , належать до множини збережених атрибутів A . Сутність оптимізації полягає в тому, що оптимізатор "проштовхує" предикат σ_c на найнижчий можливий рівень абстракції — безпосередньо до драйвера зчитування даних з диска. Таким чином, у фізичну оперативну пам'ять завантажуються виключно ті рядки та колонки, які задовольняють умови кінцевого результату.

Крім того, для оптимізації складних реляційних з'єднань, або ж операції JOIN, яка позначається як $\bowtie$, застосовується властивість дистрибутивності селекції щодо декартового добутку. Якщо предикат c обмежує виключно атрибути першого відношення R_1 , алгебраїчна трансформація набуває вигляду:

$$\sigma_c(R_1 \bowtie R_2) \equiv \sigma_c(R_1) \bowtie R_2.$$

Завдяки цим математичним трансформаціям графа, обсяг проміжних даних зменшується на порядки. Це усуває проблему переповнення оперативної пам'яті під час формування хеш-таблиць для операцій з'єднання або агрегації та максимізує загальну стабільність обчислювального конвеєра [9, 10].

2.1.3 Моделювання паралельного виконання за алгоритмом Work-Stealing

Теоретична масштабованість системи при розпаралелюванні аналітичних завдань жорстко обмежена математичним законом Амдала. Цей закон визначає верхню межу теоретичного прискорення виконання програми при використанні P багатопотокових ядер процесора. Математична формалізація закону Амдала має наступний вигляд:

$$S(P) = \frac{1}{(1 - p) + \frac{p}{P}},$$

де $S(P)$ — загальний коефіцієнт прискорення системи,

p — частка алгоритмічного коду, яка є ідеально паралелізованою,

$(1 - p)$ — частка строго послідовного коду, яка не піддається розпаралелюванню, наприклад, операції блокування, синхронізації потоків або остаточного злиття результатів [17].

Аналіз функції $S(P)$ вказує на те, що навіть за умови наявності нескінченної кількості процесорних ядер ($\lim_{P \rightarrow \infty} \frac{p}{P} = 0$), загальне прискорення системи асимптотично наближається до величини $\frac{1}{1-p}$. Це означає, що неефективне управління потоками та надмірні затримки на синхронізацію можуть повністю нівелювати переваги багатоядерного процесора.

Для наближення реальної продуктивності до теоретичного максимуму застосовується алгоритм динамічного розподілу навантаження Work-Stealing. На відміну від статичного розподілу, де кожному потоку заздалегідь виділяється фіксований обсяг даних, Work-Stealing імплементує децентралізовану модель управління. Кожен робочий потік P_i підтримує локальну двосторонню чергу завдань. Потік завжди витягує нові мікрозавдання з кінця своєї власної черги за принципом last in first out, що забезпечує мінімальні витрати на синхронізацію та зберігає просторову

локальність кешу [18].

Якщо ж локальна черга потоку P_i стає порожньою, він перетворюється на "крадія". Потік-крадій випадковим чином обирає інший активний потік P_j і "викрадає" завдання з початку його черги за принципом first in first out. Математично доведено, що час виконання паралельного графа обчислень за алгоритмом Work-Stealing (T_p) має жорстку верхню межу:

$$E \leq \frac{T_1}{P} + O(T_\infty),$$

де: T_1 — загальний обсяг роботи, час виконання завдання на одному процесорному ядрі;

T_∞ — довжина критичного шляху, мінімальний час виконання за наявності нескінченної кількості ядер;

P — кількість фізичних процесорів.

За умови, що загальний обсяг розпаралелюваної роботи значно перевищує довжину критичного шляху ($T_1 \gg T_\infty$), час виконання асимптотично наближається до оптимального значення $\frac{T_1}{P}$, що підтверджує високу масштабованість системи при обробці мікро-батчів [10].

2.2 Математичне моделювання поведінкових патернів та метрик фрустрації

Аналіз неструктурованих часових рядів сирих подій вимагає алгоритмічної трансформації дискретних міток у безперервні математичні показники. Відповідно, для виявлення прихованих закономірностей у поведінці користувача та вимірювання його задоволеності або фрустрації, застосовуються спеціалізовані метрики, що базуються на теорії ймовірностей та теорії інформації.

2.2.1 Формалізація індексів успішності сценаріїв

Task Success Rate (TSR) — це базова поведінкова метрика результативності, яка математично визначає ймовірність того, що взаємодія користувача з певним функціональним модулем закінчиться досягненням кінцевої мети. У контексті предиктивної аналітики стабільний та високий показник TSR корелює зі зниженням тертя в інтерфейсі.

Для математичної формалізації TSR введемо загальний простір подій E , що згенеровані користувачем у межах аналізованої сесії S . Множина всіх зафіксованих подій-тригерів, які ініціюють початок завдання, визначається як підмножина $E_{attempt} \subset E$. Множина подій, що однозначно підтверджують успішне завершення цього завдання, визначається як $E_{success} \subset E$.

Значення TSR розраховується як математичне відношення кількості успішних цільових дій до кількості спроб. Для обчислення використовується індикаторна функція $\mathbb{I}$, яка набуває значення 1, якщо подія e_i належить до цільової підмножини, і 0 — в іншому випадку:

$$TSR = \frac{\sum_{i=1}^N \mathbb{I}(e_i \in E_{success})}{\sum_{i=1}^N \mathbb{I}(e_i \in E_{attempt})}, \quad (2.3)$$

де N — загальна кількість подій у сесії. Показник TSR формує значення у замкненому діапазоні. Відхилення TSR окремого користувача від медіанних значень є одним із найпотужніших нелінійних предикторів ймовірності відмови від подальшої конверсії.

2.2.2 Математична оцінка когнітивного зусилля та фрустрації

Визначення рівня фрустрації неможливе без кількісної оцінки відхилень поведінки індивіда від статистичної норми когорти. Для локалізації

аномальних витрат часу на виконання стандартних мікро-завдань застосовується розрахунок Z -оцінки стандартного відхилення. Формула Z -оцінки для часу перебування на специфічному екрані t має вигляд:

$$Z_t = \frac{t - \mu_{cohort}}{\sigma_{cohort}},$$

де: μ_{cohort} — середнє арифметичне значення часу перебування для певної когорти користувачів;

σ_{cohort} — середньоквадратичне відхилення.

Значення $Z_t > 2$ або $Z_t < -2$ свідчить про наявність статистичного викиду, що інтерпретується як затримка через когнітивне перевантаження або технічну помилку [19].

Більш глибокою метрикою фрустрації є індекс загубленості або оцінка когнітивного зусилля — Cognitive Effort Score, CES. В основі цієї метрики лежить гіпотеза про те, що впевнена поведінка є лінійною і не містить надлишкових циклів. Якщо користувач регулярно стикається з нерозумінням інтерфейсу, його шлях стає циклічним, наприклад, переходи $A \rightarrow B \rightarrow A \rightarrow C \rightarrow A$.

Математично CES формалізується як відношення загального обсягу виконаних дій до кількості унікальних функціональних вузлів, які були задіяні для розв'язання завдання:

$$CES = \frac{\sum_{j=1}^S N_j}{\sum_{j=1}^S U_j}, \quad (2.4)$$

де: S — множина, яка позначає релевантні сесії користувача;

N_j — загальна кількість усіх згенерованих подій у j -й сесії;

U_j — кількість унікальних категорій дій.

Оптимальний сценарій використання передбачає наближення значення CES до 1. Значне перевищення одиниці ($CES \gg 1$) є математичним доказом генерування надлишкових дій та беззаперечним симптомом високого рівня фрустрації [3].

2.2.3 Інформаційно-теоретична оцінка хаотичності дій

Хаотичність та різноманітність поведінки користувача можуть бути кількісно оцінені за допомогою інструментів теорії інформації. Якщо користувач точно знає свою мету, він взаємодіє з обмеженою кількістю категорій інтерфейсу. Якщо ж його мета розмита і він відчуває фрустрацію, він генерує рівномірний розподіл подій по багатьох категоріях.

Використовуючи класичну формулу ентропії Шеннона, можливо розрахувати ступінь хаотичності сесії користувача. Нехай уся множина подій сесії розподілена за k дискретними категоріями дій, наприклад, search, settings, push, error. Ентропія дій $H(A)$ розраховується як:

$$H(A) = - \sum_{i=1}^k p(a_i) \log_2 p(a_i), \quad (2.5)$$

де $p(a_i)$ — емпірична ймовірність виникнення події i -ї категорії в межах однієї сесії.

Низька ентропія $H(A) \rightarrow 0$ означає, що користувач концентрується на виконанні одноманітних, монотонних дій. Висока ентропія свідчить про розпорошеність уваги та непередбачуваність поведінки, що є прямим індикатором сценаріїв з високим когнітивним навантаженням під час етапу онбордингу [20].

2.3 Теоретичні засади векторизації та трансформації ознак

Використання базових статистичних агрегацій, тобто кількості подій або середніх значень, має фундаментальний недолік — повну втрату інформації про хронологічний порядок взаємодій. Для захоплення послідовної природи поведінки застосовуються інструменти обробки природної мови — NLP.

2.3.1 Статистичне обґрунтування метрики TF-IDF для поведінкових послідовностей

Адаптація методів NLP до аналізу логів передбачає розгляд історії дій користувача як своєрідного текстового "документа" d . Окремі дискретні події або їх комбінації, зафіксовані в системі, виступають у ролі "термінів" t . Відповідно, повна сукупність профілів усіх користувачів когорти формує "корпус" D .

Алгоритм зважування TF-IDF дозволяє елегантно розв'язати проблему домінування рутинних подій. Він привласнює кожній події вагу, що прямо пропорційна її інтенсивності в межах конкретної сесії, але логарифмічно штрафує ті події, які масово присутні у поведінці всієї когорти, наприклад, `app_open`. Математичний розклад метрики складається з двох множників.

Частота терміна TF вимірює локальну значущість події з урахуванням згладжування, щоб уникнути зміщення через різну тривалість сесій:

$$TF(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

де: $f_{t,d}$ — абсолютна кількість виникнень події t у профілі користувача d ;

$\sum_{t' \in d} f_{t',d}$ — загальна кількість усіх подій у цій сесії [11].

Обернена частота документа IDF математично вимірює глобальну інформативність події:

$$IDF(t, D) = \log \frac{|D|}{|\{d \in D: t \in d\}| + 1},$$

де $|D|$ — загальна кількість користувачів у когорті. Знаменник відображає кількість користувачів, які згенерували подію t хоча б один раз. Додавання одиниці до знаменника, тобто логарифмічне згладжування, є обов'язковою математичною практикою для унеможливлення помилки ділення на нуль [11].

Фінальна вага терміна в розрідженій матриці визначається добутком:

$$TFIDF(t, d, D) = TF(t, d) \times IDF(t, D), \quad (2.6)$$

2.3.2 Контекстуалізація через n-грами та динамічне зважування

Застосування виключно уніграм, або ж ізольованих подій, не дозволяє моделі розрізняти хронологічну спрямованість патернів, оскільки за уніграмною логікою послідовність подій $A \rightarrow B$ ідентична послідовності $B \rightarrow A$. У контексті аналізу конверсії різниця є суттєвою. Для збереження інформації про транзакційні переходи конвеєр здійснює алгоритмічну генерацію n-грамів — безперервних послідовностей подій довжини n , зазвичай біграм та триграм.

Однак застосування n-грамів породжує проблему "прокляття розмірності": кількість унікальних комбінацій зростає експоненційно зі збільшенням довжини n , перетворюючи ознаковий простір на гіпервимірну, екстремально розріджену матрицю.

Для редукції розмірності та стабілізації процесу машинного навчання застосовується фільтрація ознак за математичним порогом мінімальної частоти документа $\min_df$. Алгоритм векторизації ігнорує будь-яку n-граму, для якої виконується умова:

$$\frac{|\{d \in D : t \in d\}|}{|D|} < \min df, \quad (2.7)$$

Цей механізм гарантовано відсікає статистичний шум, наприклад, унікальні збої застосунку або індивідуальні нетипові кліки, запобігаючи швидкому перенавчанню моделі на артефактах вибірки.

2.3.3 Математичні методи стабілізації дисперсії

Динамічні часові показники поведінки користувачів, такі як тривалість сесій, кількість звернень до модулів, характеризуються асиметричними розподілами з важкими хвостами. Наявність користувачів із аномально високою активністю ускладнює побудову оптимальних гістограм в алгоритмах машинного навчання.

Для математичної стабілізації дисперсії реалізовано двоступеневий процес. Першим кроком є нелінійне просторове стиснення шляхом вінзоризації — P99 Clipping. Значення x_i для кожного користувача примусово обмежується 99-м процентилем $P_{0.99}(X)$ кумулятивної функції розподілу популяції:

$$x_i^* = \min(x_i, P_{0.99}(X)). \quad (2.8)$$

операція нейтралізує деструктивний вплив аномальних значень, захищаючи градієнт функції втрат від "вибухів" під час оптимізації.

Другим кроком застосовується зсунута логарифмічна трансформація Log1p. Використання класичного натурального логарифма $\ln x$ є математично неприпустимим, оскільки $\ln 0$ генерує невизначеність, а розріджені матриці переважно складаються з нулів. Зсув аргументу на одиницю вирішує цю проблему:

$$x' = \ln(1 + x_i^*). \quad (2.9)$$

рансформація Log1p дозволяє лінеаризувати експоненційні залежності. Вона базується на законі спадної граничної корисності: різниця між 0 та 5 помилками має колосальний негативний вплив на конверсію, тоді як різниця між 100 та 105 помилками логарифмічно згладжується, допомагаючи моделі знаходити кращі пороги розщеплення.

Хоча класичні дерева рішень є інваріантними до монотонних перетворень і здатні знаходити точки розрізу незалежно від масштабу, сучасні алгоритми, зокрема LightGBM та XGBoost, використовують гістограмну

оптимізацію, розбиваючи неперервні ознаки на фіксовану кількість кошиків перед початком навчання. Для сильно скошених розподілів логарифмування гарантує, що ці кошики будуть розподілені рівномірно, запобігаючи злиттю більшості нормальних спостережень в один бакет через кілька аномальних значень на краю розподілу.

2.4 Алгоритмічне обґрунтування методів градієнтного бустингу

Згенерований багатовимірний простір, що складається зі щільних поведінкових метрик та розріджених NLP-матриць, є ідеальним базисом для сімейства алгоритмів градієнтного бустингу над деревами рішень — Gradient Boosting Decision Trees, GBDT. Ці моделі доведено забезпечують найвищу точність на гетерогенних табличних даних та стійкість до дисбалансу класів.

2.4.1 Математична постановка задачі адитивного навчання

Теорія градієнтного бустингу спирається на концепцію послідовного адитивного навчання ансамблю слабких учнів — дерев рішень. Прогноз моделі на ітерації t для i -го екземпляра $\hat{y}_i^{(t)}$ формується як алгебраїчна сума прогнозу попереднього кроку та виводу нової базової функції f_t :

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i).$$

Для оптимізації алгоритм мінімізує загальну цільову функцію $L^{(t)}$, яка складається з диференційованої функції втрат l , яка у задачах класифікації застосовується LogLoss, та структурного регуляризаційного члена Ω , який відповідає за запобігання перенавчанню:

$$L^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t).$$

Регуляризація $\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$ штрафує модель за надмірну глибину дерева T — кількість листків, та великі значення ваг w у листках [21].

2.4.2 XGBoost: оптимізація другого порядку Тейлора

Архітектура XGBoost відрізняється від класичного бустингу використанням математичної апроксимації цільової функції за допомогою ряду Тейлора другого порядку. Це дозволяє враховувати не лише напрямок градієнтного спуску, але й кривизну функції втрат, забезпечуючи швидшу збіжність алгоритму. Розклад функції має вигляд:

$$L^{(t)} \approx \sum_{i=1}^n \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t),$$

де: $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ — перша похідна;

$h_i = \partial_{\hat{y}_i^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$ — друга похідна функції втрат.

Фундаментальною оптимізацією XGBoost для роботи з розрідженими матрицями, зокрема TF-IDF, є алгоритм Sparsity-aware split finding. Під час пошуку оптимального порогу розщеплення, алгоритм алгоритмічно направляє всі екземпляри з пропущеними або нульовими значеннями у дефолтну гілку, яка обчислюється динамічно за критерієм найбільшого зниження функції втрат, кардинально зменшуючи кількість математичних операцій [13].

2.4.3 LightGBM: градієнтний однобічний семплінг та об'єднання ознак

Специфіка фреймворку LightGBM полягає у вирішенні проблеми обчислювальної складності під час обробки надвеликих масивів даних шляхом застосування двох інновацій: Gradient-based One-Side Sampling (GOSS) та Exclusive Feature Bundling (EFB).

Алгоритм GOSS спирається на властивість того, що зразки даних із великими градієнтами, екземпляри, де модель сильно помиляється, несуть у собі найбільший потенціал для приросту інформації. GOSS сортує вибірку за градієнтом, повністю зберігаючи топ $a \cdot 100\%$ екземплярів із найбільшими помилками. З решти екземплярів береться випадкова вибірка $b \cdot 100\%$. Для того, щоб не спотворити математичне очікування дисперсійного приросту, ваги градієнтів для випадково обраної підмножини з малими помилками підсилюються на коефіцієнт $\frac{1-a}{b}$. Формула приросту інформації $V_j(d)$ для розщеплення за ознакою j в точці d модифікується наступним чином:

$$V_j(d) = \frac{1}{n} \left(\frac{\left(\sum_{x_i \in A_l} g_i + \frac{1-a}{b} \sum_{x_i \in B_l} g_i \right)^2}{n_l(d)} + \frac{\left(\sum_{x_i \in A_r} g_i + \frac{1-a}{b} \sum_{x_i \in B_r} g_i \right)^2}{n_r(d)} \right),$$

де: A — множина з великими градієнтами;

B — рандомізована множина з малими градієнтами;

l і r — лівий та правий вузли після розщеплення.

Концепція EFB зменшує розмірність розріджених даних шляхом об'єднання взаємовиключних ознак. Оскільки більшість колонок TF-IDF ніколи не набувають ненульових значень одночасно для одного запису, LightGBM зводить процес групування ознак до математичної задачі розфарбовування графа. Через використання жадібного алгоритму, розріджені

ознаки зливаються у щільні гістограмні бандли, знижуючи часову складність навчання від $O(N_{data} \times N_{features})$ до $O(N_{data} \times N_{bundles})$ [15].

2.4.4 CatBoost: впорядковані цільові статистики та симетричні дерева

Робота з гетерогенними категоріальними ознаками без використання екстенсивного One-Hot Encoding здійснюється за допомогою архітектури CatBoost. Її головною перевагою є механізм OTS, який вирішує математичну проблему витоку цільової змінної у тренувальну вибірку.

Класичне цільове кодування несе в собі ризик того, що мітка самого об'єкта впливатиме на розрахунок його ж ознаки. Щоб уникнути цього, CatBoost запроваджує штучну вісь часу, використовуючи випадкові перестановки вибірки σ . Обчислення статистики для i -го екземпляра в перестановці базується виключно на спостереженнях, які логічно йому передують ($j < i$). Формула OTS має вигляд:

$$OTS_i = \frac{\sum_{j=1}^{i-1} \|(x_{\sigma(j)} = x_{\sigma(i)}) \cdot y_{\sigma(j)} + a \cdot P}{\sum_{j=1}^{i-1} \|(x_{\sigma(j)} = x_{\sigma(i)}) + a},$$

де: $\|$ — індикаторна функція, яка фіксує збіг значень категорій;

y — цільова змінна;

P — апріорна ймовірність;

a — коефіцієнт згладжування, що запобігає дисперсійним викидам на початку перестановки.

Також CatBoost застосовує геометрію симетричних дерев — на кожному рівні глибини використовується абсолютно однаковий предикат розщеплення для всіх існуючих вузлів. Це слугує потужним регуляризатором і кардинально прискорює етап інференсу за допомогою ефективних побітових операцій [14].

Порівняльна характеристика відмінностей математичного апарату в

алгоритмах GBDT наведена в таблиці 2.2.

Таблиця 2.2 — Фундаментальні відмінності математичного апарату в алгоритмах GBDT

Алгоритм	Основна математична оптимізація	Механізм боротьби з розрідженістю	Стратегія розвитку дерева
XGBoost	Апроксимація Тейлора 2-го порядку	Sparsity-aware split finding	Зростання по рівнях
LightGBM	Семплінг за градієнтом GOSS (V_j переважування)	Злиття графів ознак EFB	Зростання по листках
CatBoost	Обчислення цільових статистик OTS через перестановки σ	Комбінаторика та злиття категорій	Симетрична топологія

2.4.5 Математичне обґрунтування валідації в часі

У класичному машинному навчанні стандартним методом оцінки узагальнюючої здатності алгоритмів є K-Fold крос-валідація. Математично вона передбачає рівномірне та випадкове розбиття початкової множини даних на K підмножин однакового розміру. На кожній ітерації $k - 1$ фолдів використовуються для навчання, а один — для тестування. Проте в задачах предиктивної поведінкової аналітики, де спостереження є послідовностями логів дій із вираженою темпоральною природою, використання випадкового K-Fold розбиття є фундаментально некоректним.

Випадкове семплювання індексів масиву ігнорує часову залежність

даних. Це неминуче призводить до перехресного витоку даних — Data Leakage, а в контексті часових рядів — Look-ahead bias. За такої парадигми виникають ситуації, коли модель під час навчання отримує доступ до подій, що відбулися хронологічно пізніше за події, які вона має передбачити у валідаційній вибірці. Таким чином, алгоритм машинного навчання використовує знання з "майбутнього" для класифікації "минулого". Це грубо порушує причинно-наслідковий зв'язок і призводить до ілюзії генералізації, тобто переоцінки реальної точності моделі на етапі розробки [22].

Для усунення цієї структурної проблеми в системі конвеєрної аналітики застосовується метод валідації на основі хронологічного розрізу, який в академічній літературі також відомий як оцінка прогнозу з рухомим початком або Out-of-Time крос-валідація. Цей метод базується на збереженні строгого хронологічного порядку всіх спостережень.

Нехай $D = \{e_1, e_2, \dots, e_N\}$ — повна множина історичних подій, суворо впорядкована за часовими мітками так, що $t_1 \leq t_2 \leq \dots \leq t_N$. Під час крос-валідації на K кроках для кожної ітерації k , де $k \in \{1, 2, \dots, K\}$, алгоритм генерує дві непересічні множини: навчальну вибірку $Train_k$ та валідаційну вибірку Val_k .

Фундаментальна аксіома коректності такого розбиття полягає у тому, що максимальна часова мітка в навчальній множині повинна бути строго меншою за мінімальну часову мітку у валідаційній:

$$\forall t_i \in Train_k, \forall t_j \in Val_k \Rightarrow t_i < t_j.$$

Практична алгоритмічна реалізація цього математичного обмеження зазвичай виконується за двома сценаріями.

1. Метод вікна, що розширюється, коли на кожному кроці k межа розбиття, або поточна точка спостереження T_k , зміщується вперед у часі. Навчальна вибірка постійно зростає, акумулюючи всі доступні історичні дані з минулого, тоді як тестування відбувається на наступному фіксованому проміжку часу: $Train_k = \{e_i \in D \mid t_i \leq T_k\}$, $Val_k = \{e_i \in D \mid T_k < t_j \leq T_k + \Delta t\}$, де Δt — заданий горизонт

прогнозування, наприклад, наступні 7 днів.

2. Метод рухомого вікна, коли розмір навчальної вибірки жорстко фіксується за допомогою параметра W , наприклад, історія рівно за останні 30 днів, і це "вікно" синхронно ковзає вздовж осі часу разом із валідаційним горизонтом. Цей підхід є оптимальним для додатків, де поведінкові патерни користувачів швидко еволюціонують або застарівають: $Train_k = \{e_i \in D \mid T_k - W < t_i \leq T_k\}$.

Застосування стратегії Time-Series Split гарантує, що для будь-якої ітерації k алгоритм градієнтного бустингу розраховує градієнти, будує дерева та оптимізує свої ваги виключно на основі того масиву інформації, який був би доступний системі на момент часу T_k у реальних умовах експлуатації. Такий підхід повністю імітує production-середовище та забезпечує неупереджену, ізольовану оцінку метрик на дійсно невідомих, "майбутніх" даних [23].

2.5 Математичне обґрунтування інтерпретованості та калібрування результатів

Використання високоточних нелінійних моделей породжує проблему "чорної скриньки", що унеможливорює безпосереднє застосування результатів у бізнес-логіці без детальної інтерпретації прийнятих рішень та математичного калібрування порогів відсічення.

2.5.1 Теорія ігрових значень Шеплі

Стандартні вбудовані метрики важливості ознак у деревах рішень мають доведене математичне зміщення на користь неперервних змінних або категоріальних ознак із великою кількістю унікальних значень. Алгоритм частіше обирає їх для розщеплення просто через більшу кількість доступних точок розрізу, що спотворює реальну картину впливу ознак.

Для прозорості та справедливого пояснення маргінального впливу кожної ознаки застосовується методологія SHAP, SHapley Additive exPlanations [24, 35]. Вона позбавлена цього зміщення, оскільки базується на кооперативній теорії ігор. Математичний апарат SHAP заснований на кооперативній теорії ігор: процес генерації прогнозу розглядається як спільна "гра", де ознаки виступають "гравцями", а сам прогноз — "виграшем", який необхідно справедливо розподілити. Математичне визначення значення Шеплі ϕ_j для j -ї ознаки розраховується як зважене середнє маргінального внеску цієї ознаки по всіх можливих комбінаціях підмножин S — коаліціях ознак:

$$\phi_j(f, x) = \sum_{S \subseteq F \setminus \{j\}} \frac{|S|! (|F| - |S| - 1)!}{|F|!} (f_x(S \cup \{j\}) - f_x(S)),$$

де: F — загальна множина ознак;

$|F|$ — їх кількість;

$f_x(S)$ — математичне очікування прогнозу моделі, коли відомі лише значення ознак із множини S .

Унікальність цього методу полягає в тому, що він єдиний строго задовольняє чотири математичні аксіоми чесного розподілу. Першою з них є ефективність (або локальна точність), яка гарантує, що сума впливів усіх ознак завжди дорівнює різниці між фактичним прогнозом для конкретного екземпляра $\hat{f}(x)$ та базовим математичним очікуванням $E[\hat{f}(X)]$ усієї вибірки. Математично це виражається рівнянням:

$$\sum_{j=1}^{|F|} \phi_j = \hat{f}(x) - E[\hat{f}(X)].$$

Друга аксіома – симетрія, яка визначає, що якщо дві ознаки j та k роблять абсолютно однаковий внесок у будь-яку з можливих коаліцій S , вони отримують ідентичні значення Шеплі ($\phi_j = \phi_k$). Третя властивість, фіктивність, вказує на те, що ознака, яка жодним чином не змінює прогноз при додаванні до будь-якої коаліції, має нульове значення впливу ($\phi_j = 0$). Нарешті, четвертою аксіомою є адитивність, яка забезпечує консистентність для ансамблевих алгоритмів. Згідно з нею, загальний вплив ознаки у композиції моделей, таких як випадкові ліси чи градієнтні бустинги, строго дорівнює алгебраїчній сумі її впливів у кожному окремому базовому дереві.

Шляхом застосування SHAP, система здатна трансформувати абстрактну ймовірність у зрозумілий вектор тригерів фрустрації чи конверсії [24, 35].

2.5.2 Математична оптимізація порогу прийняття рішень для маркетингових сигналів

Агресивне впровадження політик конфіденційності, таких як App Tracking Transparency від Apple в iOS 14.5, а також блокування сторонніх файлів cookie у сучасних браузерях, спровокували тектонічний зсув в індустрії: колосальну втрату детермінованих сигналів зворотного зв'язку. У результаті алгоритми соціальних мереж втрачають здатність відстежувати користувачів, які здійснили конверсію через кілька днів після натискання на рекламу. Рекламні кампанії починають діяти наосліп, що призводить до стрімкого падіння рентабельності рекламних витрат та залучення неякісного трафіку [25].

Єдиним технічним рішенням для подолання цього бар'єра є архітектурний перехід до Server-to-Server комунікації через протоколи типу САРІ. Замість очікування запізнілих та заблокованих подій від клієнтських пристроїв, бекенд мобільного застосунку інтегрує розроблену модель машинного навчання для генерації так званих "синтетичних конверсій". Цей процес вимагає безпрецедентної уваги до математики прийняття рішень, а саме — до калібрування порогу відсічення [16].

Концепція синтетичних конверсій базується на перенесенні сигналу з самого низу воронки продажів, тобто факту списання коштів за підписку після завершення тріалу, на її верхню або середню частину — перші 24 години після інсталяції, за допомогою предиктивного моделювання.

Архітектура цього процесу працює наступним чином: серверна інфраструктура збирає сирі логи перших сесій нового користувача, трансформує їх у багатовимірні вектори і подає на інференс моделі машинного навчання. Алгоритм миттєво розраховує безперервну ймовірність того, що цей користувач оформить платну підписку. Якщо ця ймовірність перетинає заздалегідь встановлений поріг відсічення, система приймає детерміноване рішення класифікувати цього користувача як "Високоймовірного покупця".

Не очікуючи фактичного завершення транзакції, бекенд формує захищений HTTP POST-запит і відправляє його безпосередньо до кінцевих точок Meta Conversions API. У цьому пакеті даних синтетична подія маркується як віртуальний Purchase або HighValue_Trial із розрахованою прогнозованою цінністю життєвого циклу. Внутрішні рекламні алгоритми машинного навчання, отримавши цей ранній, чистий та безперебійний серверний сигнал, миттєво коригують ваги аукціону, знаходячи закономірності між профілем цього синтетичного покупця та іншою аудиторією мережі, що дозволяє таргетувати найбільш релевантних користувачів [16].

Великим інженерним викликом у реалізації цієї архітектури є вибір правильного порогу відсічення ймовірності. Традиційні алгоритми бінарної

класифікації за замовчуванням використовують симетричний поріг 0.5: якщо $P > 0.5$, об'єкту присвоюється клас 1, якщо ні — клас 0. У задачах мобільної аналітики, де спостерігається екстремальний дисбаланс класів, дефолтний поріг 0.5 є математично хибним і призводить до повного ігнорування міноритарного класу. Для вирішення цієї проблеми застосовується стратегія "зміщення порогу".

У контексті відправки синтетичних сигналів через САРІ до рекламної мережі, необхідно проаналізувати наслідки роботи порогу через класичну матрицю плутанини, перевірши статистичні помилки у площину витрат на цифровий маркетинг:

1. Істинно позитивні (TP) — модель правильно ідентифікувала майбутнього покупця. Сигнал відправляється у Meta. Алгоритм успішно навчається на правильному еталонному патерні.
2. Істинно негативні (TN) — модель правильно проігнорувала користувача без наміру платити. Рекламний бюджет зберігається від розтрати.
3. Хибнонегативні (FN) — модель не розпізнала реального покупця. Це призводить до падіння метрики Повноти. Бізнес втрачає можливість навчити рекламну мережу на цьому конкретному користувачі, проте такий пропуск не завдає прямої шкоди алгоритму оптимізатора.
4. Хибнопозитивні (FP) — модель помилково класифікувала випадкового користувача як покупця. Відправка такого сигналу через САРІ дезінформує алгоритми Meta. Рекламна система сприймає цей сигнал як еталон і починає шукати "схожих" користувачів на цього нерелевантного суб'єкта, що призводить до експоненційного витрачання рекламних бюджетів на нецільову аудиторію.

З огляду на цю бізнес-логіку, математична оптимізація порогу відсічення для синтетичних конверсій має бути суворо асиметричною: Точність має абсолютний пріоритет над Повнотою. Показник точності повинен підтримуватися на максимально можливому рівні, щоб гарантувати бездоганну якість синтетичних лідів та захистити рекламну мережу від

інформаційного отруєння.

Будь-який алгоритм класифікації генерує на виході умовну ймовірність $P(y = 1 | x)$ конверсії. Традиційно застосовується дефолтний поріг прийняття рішення $\tau = 0.5$. Проте у задачах мобільної поведінкової аналітики спостерігається екстремальний дисбаланс класів: успішна покупка є статистично рідкісною подією у порівнянні з відтоком.

З точки зору Баєсівської ймовірності, якщо апіорна ймовірність конверсії у популяції становить, наприклад, 5%, то ідеально відкалібрована модель буде генерувати середню ймовірність по датасету також на рівні 0.05. За таких умов стандартний поріг відсічення $\tau = 0.5$ є математично недосяжним для абсолютної більшості реальних користувачів. Тому зсув порогу у задачах з дисбалансом класів є математичною необхідністю.

Для автоматизації процесу передачі сигналів, наприклад, для CAPI, необхідно обчислити оптимальний поріг τ , який максимізує баланс бізнес-метрик. Для цього розраховується гармонійне середнє (F1-score), яке вимагає попереднього визначення Точності (Precision) та Повноти (Recall).

Точність визначає частку істинно позитивних результатів серед усіх об'єктів, які алгоритм класифікував як позитивні:

$$Precision(\tau) = \frac{TP(\tau)}{TP(\tau) + FP(\tau)}.$$

Повнота визначає частку істинно позитивних результатів, які модель змогла ідентифікувати серед усіх фактично позитивних об'єктів у вибірці:

$$Recall(\tau) = \frac{TP(\tau)}{TP(\tau) + FN(\tau)},$$

де TP , FP , FN — відповідні елементи матриці помилок при обраному порозі τ .

Відповідно, метрика F1-score як функція від порогу має вигляд:

$$F_1(\tau) = \frac{2 \cdot Precision(\tau) \cdot Recall(\tau)}{Precision(\tau) + Recall(\tau)} = \frac{2 \cdot TP(\tau)}{2 \cdot TP(\tau) + FP(\tau) + FN(\tau)}.$$

Математично доведено низкою досліджень, що для класифікатора, який генерує ідеально калібровані через ізотонічну регресію або алгоритм Платта апіорні ймовірності, оптимальний поріг прийняття рішення, що максимізує

метрику F1-score, строго дорівнює половині від максимально досяжного значення F1-score [26]. Формальний вираз теореми має вигляд:

$$\tau_{opt} = \frac{\max(F_1)}{2}.$$

Цей фундаментальний висновок ліквідує потребу в ресурсомісткому переборі сітки порогів під час інференсу у виробничому середовищі. Динамічне зміщення порогу до τ_{opt} дозволяє системі з математичною точністю ідентифікувати ту частку користувачів, які потребують залучення, балансуючи між уникненням нецільових витрат бюджету та втратою цільової платоспроможної аудиторії [27].

2.6 Висновки до розділу 2

У рамках другого розділу створено ґрунтовний математичний та алгоритмічний апарат, що лежить в основі розробленої системи прогнозування конверсії користувачів.

Відмова від традиційної рядкової реляційної моделі на користь колонкової архітектури Apache Arrow доведено мінімізує обчислювальні витрати, оскільки звернення до пам'яті пропорційне виключно до формули 2.2. Аналіз графа обчислень в умовах лінивої оцінки дозволив оптимізатору застосувати закони реляційної алгебри — проштовхування предикатів та зрізання проєкцій, що нівелює ризики переповнення оперативної пам'яті. Межі масштабування закону Амдала було успішно розширено завдяки імплементації децентралізованого алгоритму Work-Stealing, який забезпечує виконання в межах теоретичного оптимуму.

Аналіз фрустрації та поведінки переведено із суб'єктивної площини у формальну. Запроваджено використання індикаторних функцій для розрахунку ймовірностей завершення завдання. Використано методи Z-оцінки

для виявлення статистичних викидів та індекс когнітивного зусилля, де високе співвідношення загальних дій до унікальних слугує маркером циклічної, "загубленої" поведінки. Ентропія Шеннона адаптована для виявлення розпорошеності уваги в інтерфейсі.

Векторизація логів математично забезпечується адаптацією методології NLP. Застосування TF-IDF дозволило оцінити локальну значущість івентів на фоні глобального корпусу, а інтеграція n-грам (біграм та триграм) убезпечила контекстуальну направленість дій. Прокляття розмірності успішно подолано через фільтрацію $\min_df$, а важкі хвости розподілу — через операцію зсунутої логарифмічної трансформації $\ln(1 + x)$ та процентильного вінзорування.

Теоретичний базис предиктивних алгоритмів розглянуто крізь призму цільової функції з регуляризацією в адитивному навчанні. Надано чітке обґрунтування переваг кожної моделі: апроксимація функції втрат через ряд Тейлора 2-го порядку для XGBoost; застосування семплінгу GOSS із формулою компенсації дисперсійного приросту для LightGBM; а також механізм OTS у CatBoost, що використовує перестановки вибірки для боротьби з витоком даних.

Фінальний етап інтерпретації прогнозів базується на ігрових значеннях Шеплі, що доведено задовольняють аксіомам локальної точності та симетрії. Калібрування порогу для систем автоматизованого маркетингу, таких як CAPI, спирається на доведену теорему максимізації гармонійного середнього. Створена теоретична та архітектурна база забезпечує готовність моделі до безперебійної інтеграції у промислові аналітичні платформи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ПРОГНОЗУВАННЯ КОНВЕРСІЇ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ

3.1 Інструментальні засоби та обчислювальна архітектура

Розробка промислової системи предиктивної аналітики для мобільного застосунку вимагає вирішення комплексу інженерних та математичних завдань, що виходять за межі класичного прототипування. Опрацювання хронологічних послідовностей користувацьких сесій, які містять десятки мільйонів записів, формує безпрецедентне навантаження на обчислювальну інфраструктуру. Зважаючи на те, що архітектура системи проєктується для функціонування в умовах апаратних обмежень, фундаментальним критерієм успіху є зменшення споживання оперативної пам'яті та забезпечення максимальної утилізації сучасних багатоядерних процесорів і спеціалізованих графічних прискорювачів.

Проєктування програмного комплексу базується на принципах мікросервісної організації даних, де етапи вилучення ознак, трансформації та інференсу чітко розмежовані на рівні файлової системи. Сирі логи акумулюються у директорії RAW_DATA_DIR, після чого проходять стадію збагачення та фільтрації, потрапляю чи до PROCESSED_DATA_DIR. Унікальний поведінковий контекст, трансформований у числові вектори, формує спеціалізовані сутності у FEATURE_STORE_DIR для історичних даних та FEATURE_LOGS_DIR для подій у реальному часі, що в кінцевому підсумку об'єднуються у матриці TRAINING_DATASETS_DIR для навчання алгоритмів. Зв'язування цих компонентів у єдиний безперервний цикл потребує використання найсучасніших інструментальних засобів, здатних мінімізувати накладні витрати на серіалізацію та десеріалізацію гетерогенних

структур.

3.1.1 Мова програмування та обчислювальне ядро

Основним середовищем розробки програмного комплексу виступає мова програмування Python. Відповідно до специфікації залежностей у конфігураціях проєкту, базовою вимогою є використання версії Python не нижче ≥ 3.12 . Вибір цієї ітерації інтерпретатора CPython є рішенням, спрямованим на підвищення продуктивності виконання коду в умовах інтенсивних маніпуляцій з даними.

Реліз Python 3.12 приніс архітектурні оновлення, які безпосередньо впливають на швидкість виконання високонавантажених скриптів аналітики. Ключовим нововведенням стала імплементація механізмів оптимізації вбудованих генераторів згідно з ініціативою PEP 709. У контексті розробленої системи це має критичне значення для ініціалізації багаторівневих конфігураційних словників. Наприклад, трансформація складної ієрархічної структури `event_cats`, яка містить класифікацію користувацьких подій, у плоский список `EVENT_MAPPING_DATA` відбувається без створення надлишкових одноразових об'єктів функцій. Це прискорює етап парсингу метаданих до двох разів у порівнянні з ітерацією Python 3.11.

Крім того, оновлений модуль токенизації, модернізований у рамках формалізації синтаксису f-рядків, зменшує накладні витрати процесорного часу при зчитуванні масивних статичних артефактів на величину до 64%. Покращення у системі асинхронного виконання, що забезпечують приріст продуктивності до 75%, дозволяють системі більш ефективно керувати пулом з'єднань під час взаємодії із зовнішніми базами даних через SQLAlchemy та під час вивантаження агрегованих звітів до Google BigQuery.

Попри значні синтаксичні переваги та оптимізації Python 3.12,

фундаментальною проблемою мови є блокування інтерпретатора, що унеможлиблює повноцінне паралельне виконання потоків у межах одного процесу, що є суттєвим недоліком для задач машинного навчання. Традиційний стек обробки даних, що базується на бібліотеці Pandas, яка використовує формат пам'яті NumPy, виявляється нездатним масштабуватися для масивів логів, обсяг яких вимірюється десятками гігабайтів. Pandas застосовує жадібну модель виконання обчислень, де кожна проміжна операція створює повну копію даних в оперативній пам'яті. У поєднанні з відсутністю нативної багатопотоковості, такий підхід призводить до переповнення пам'яті при виконанні комплексних об'єднань масивів або операцій групування.

Для подолання цих інженерних бар'єрів ядром обробки даних у розробленому пайплайні було обрано фреймворк Polars версії 1.33.1. Фундаментальною інновацією, імплементованою в розроблюваний конвеєр, є використання абстракції LazyFrame, що реалізує парадигму лінивих обчислень. Замість негайного виконання коду під час визначення змінної, Polars формує логічний план запиту, абстрактне синтаксичне дерево. Процес обчислень ініціюється виключно після виклику методу-термінатора `.collect()` або `.sink_parquet()`. Таке відкладене виконання дозволяє оптимізатору запитів проаналізувати весь ланцюг трансформацій від завантаження файлу до агрегації фінального результату [10].

Файли Parquet структуровані як послідовність груп рядків, кожна з яких містить статистичні метадані для кожної колонки такі як: мінімум, максимум, кількість унікальних значень та null-елементів. У розробленому пайплайні оптимізатор запитів Polars взаємодіє з цими метаданими за допомогою механізмів предиктивного й проекційного проштовхування та потокової агрегації і скидання на диск [10].

Система ідентифікації когорт вимагає фільтрації логів за критеріями `original_purchase_date`, `country` та `media_source`. Коли лінивий запит містить умову фільтрації, наприклад, вибір конкретної дати активації тріалу, оптимізатор переміщує цю операцію на найнижчий рівень безпосередньо до

драйвера зчитування файлу Parquet. Замість десеріалізації всього багатогігабайтного масиву в оперативну пам'ять із подальшим застосуванням маски, Polars аналізує статистику груп рядків на диску. Якщо діапазон значень у групі рядків не задовольняє умову предиката, ця група фізично ігнорується і не завантажується. Цей метод, відомий як "Pre-filtered Parallelism", дозволяє паралелізувати зчитування масивних історій подій із колосальною економією I/O ресурсів дискової підсистеми.

Датасет мобільного додатку містить десятки побічних колонок, проте для ініціалізації простору ознак вимагаються лише ідентифікатори з визначених конфігурацією списків `cat_cols` та `num_cols`. Оптимізатор запитів Polars автоматично переписує граф виконання таким чином, щоб з диска зчитувалися виключно ці релевантні колонки, що радикально зменшує розмірність вхідного потоку байтів.

Загальний обсяг агрегованих подій із категорій `auth`, `botanist`, `disease`, `feed` та інших для тисяч користувачів утворює матриці, які фізично не вміщуються навіть у 64 ГБ оперативної пам'яті. Для подолання цього обмеження Polars налаштовано на використання потокового рушія через ініціалізацію `.collect(engine="streaming")` або запис безпосередньо у файл через `.sink_parquet()`. У цьому режимі конвеєр фрагментує дані на мікро-батчі, пропускаючи їх через вузли обчислень. Коли обсяг проміжних хеш-таблиць при виконанні операцій `GROUP BY` досягає порогу доступної пам'яті, Polars автоматично здійснює операцію скидання, тимчасово серіалізуючи ці фрагменти на швидкісний накопичувач NVMe, що запобігає аварійному завершенню процесу з помилкою `Out-of-Memory`.

Враховуючи вимоги до побудови поведінкових профілів у жорстких часових рамках, лінива архітектура фреймворку Polars дозволяє будувати предикативні конвеєри, які здатні оперувати масивами даних, що на порядок перевищують обсяг фізично встановленої оперативної пам'яті, гарантуючи при цьому максимальну пропускну здатність та стабільність програмної реалізації [9].

3.1.2 Бібліотеки машинного навчання

Центральним аналітичним завданням розроблюваної системи є передбачення ймовірності настання цільової події `target_col = "is_converted"` на базі комплексу демографічних, фінансових та поведінкових сигналів. Для побудови адекватного мапінгу між простором ознак користувача та його схильністю до оформлення підписки використовуються алгоритми градієнтного бустингу над деревами рішень (GBDT). Конфігурація проєкту імплементує три провідні індустріальні фреймворки: LightGBM, CatBoost та XGBoost.

Специфіка вхідного простору ознак полягає у його гетерогенності. З одного боку, існують щільні числові параметри, такі як вартість інсталяції (`cri`), ціна тріалу (`trial_price`) або тривалість періоду (`period_days`). З іншого високоваріативні категоріальні ознаки рекламних кампаній (`creo_type`, `source`). Проте найбільший алгоритмічний виклик становить масив ознак, згенерований на основі методів обробки природної мови, а саме TF-IDF векторизації поведінкових сиквенцій. Налаштування `top_k = 50` для кількості топових слів/грам призводить до того, що для кожного користувача формується вектор значень TF-IDF, який є надзвичайно розрідженим. Оскільки окремо взятий користувач взаємодіє лише з мізерною часткою із понад сотнею визначених `target_events`, таких як `plant_added`, `search_initiated` або `diseases_identification_success`, абсолютна більшість елементів у цьому векторі дорівнює нулю або визначається як `null`.

Фреймворк LightGBM від компанії Microsoft виступає ключовим обчислювальним рушієм для базових тренувальних пайплайнів завдяки своїй неперевершеній швидкості обробки великомасштабних масивів. Версія 4.6.0 містить оптимізовану підтримку роботи з розрідженими даними, що дозволяє бібліотеці автоматично розпізнавати нерепрезентовані або нульові значення у TF-IDF векторах (`zero_as_missing=false` за замовчуванням), ігноруючи їх під

час обчислень градієнтів, що суттєво зменшує споживання оперативної пам'яті.

Фундаментальною відмінністю архітектури LightGBM є використання гістограмного алгоритму для пошуку оптимальних точок розщеплення. Традиційні алгоритми вимагають сортування всіх значень ознаки, що має обчислювальну складність $O(\text{data})$, де data — це кількість записів. Натомість LightGBM групує безперервні значення у дискретні біни на етапі ініціалізації, зводячи складність до $O(\text{bins})$, де bins — це кількість бінів, яка зазвичай не перевищує 255. Під час побудови дерев рішень LightGBM застосовує стратегію поглиблення по листках. Алгоритм не намагається збалансувати дерево на кожному рівні, а постійно шукає листок із максимальним потенціалом зменшення функції втрат L2 loss або logloss і розщеплює саме його. У контексті мобільної аналітики, де певні мікро-патерни поведінки, наприклад, перехід `permission_request_completed` → `onboarding_skip`, можуть мати диспропорційно високий вплив на конверсію, стратегія leaf-wise здатна краще виявляти ці складні залежності. Однак така геометрія дерева підвищує ризик перенавчання, що вимагає ретельного налаштування параметрів обмеження, таких як `max_depth` та `min_data_per_group` [15].

Бібліотека CatBoost займає у пайплайні нішу експертної моделі для роботи з комбінаторикою складних нечислових ознак. Конфігурація проєкту передбачає генерацію статичних взаємодій, таких як ("`country`", "`device_type`") або ("`device_type`", "`source`"). Ці перехресні ознаки формують простори з гігантською кардинальністю. Використання класичного One-Hot Encoding для таких даних призвело б до створення сотень тисяч нових бінарних стовпців, що миттєво вичерпало б ресурси оперативної пам'яті.

CatBoost розв'язує цю проблему завдяки вбудованому механізму Ordered Target Statistics (OTS). Алгоритм динамічно розраховує середнє значення цільової змінної для кожної категорії на основі штучних перестановок навчальної вибірки, ефективно замінюючи категорію на безперервне число без витоку майбутньої інформації, який характерний для стандартного цільового

кодування. Окрім цього, CatBoost використовує парадигму симетричних дерев, тобто на одному рівні глибини дерева застосовується абсолютно однаковий предикат розщеплення для всіх існуючих вузлів. Це діє як потужний вбудований регуляризатор, який запобігає запам'ятовуванню моделлю статистичного шуму у логах, викликаного, наприклад, аномальним спрацюванням події `app_error` або багаторазовим викликом `search_initiated`. Завдяки симетричній структурі алгоритм забезпечує високу швидкість обчислень під час генерації прогнозу на нових даних, що є важливим параметром для розгортання моделі у production-середовищі та її синхронізації з маркетинговими платформами [14].

Архітектура бібліотеки eXtreme Gradient Boosting (XGBoost) базується на алгоритмі зростання дерев по рівнях, де розгалуження відбувається симетрично по всьому рівню. Важливою особливістю XGBoost у розв'язанні задачі передбачення конверсії є його алгоритм "Sparsity-aware split finding". Під час побудови дерева модель автоматично вивчає, у яку гілку спрямовувати об'єкти з нульовими значеннями TF-IDF ознак. Якщо факт відсутності користувацької взаємодії з функцією `watering_calculator` несе в собі сигнал про низьку залученість, XGBoost алгоритмічно обчислює градієнт для цього нульового стану і призначає оптимальний маршрут у дереві рішень. Попри високу надійність, XGBoost демонструє вищі вимоги до споживання оперативної пам'яті під час навчання у порівнянні з LightGBM, що обмежує його застосування для найбільш масштабних когортних аналізів у цьому пайплайні [13].

3.1.3 Середовище розробки та апаратне забезпечення

Обчислювальна складність задач, що передбачають процесинг гігабайтів сирих аналітичних подій, генерацію TF-IDF матриць за адаптованим

лексиконом та ітеративне навчання ансамблів градієнтного бустингу, висуває вимогу щодо високопродуктивного апаратного забезпечення. Весь цикл розробки, від зневадження мікро-батчів до повномасштабних тренувальних епох, виконувався у віддаленому режимі за протоколом SSH з використанням середовища розробки Visual Studio Code, що дозволяє зберігати гнучкість локального написання коду, повністю покладаючись на ресурси спеціалізованої обчислювальної машини.

Керування залежностями та змінними середовища, організовано за допомогою пакету `python-dotenv`, що забезпечує ізоляцію конфігурацій та безпечне зберігання кренденціалів. Ініціалізація відносних шляхів та структури директорій уніфікована за допомогою бібліотеки `rootutils`, що гарантує відтворюваність експериментів незалежно від точки входу у скрипт.

Апаратна конфігурація сервера була специфікована з урахуванням вимог машинного навчання: подолання вузьких місць між оперативною та кеш-пам'яттю, забезпечення паралелізації на рівні інструкцій та підтримки тензорних обчислень.

Ключовим обчислювальним компонентом системи виступає мікропроцесор AMD Ryzen 9 5950X. Архітектурно цей процесор належить до покоління Zen 3 та містить 16 фізичних ядер, здатних одночасно виконувати 32 потоки інструкцій (SMT). Для аналітичного стеку, що базується на Polars, багатопотоковість є вирішальним фактором, адже під час виконання операцій групування Polars автоматично розбиває навантаження на всі 32 потоки. Однак, найбільш вибагливим параметром цього процесора для задач інженерії даних є масивна уніфікована кеш-пам'ять третього рівня (L3 Cache) обсягом 64 мегабайти. Робота з величезними таблицями постійно супроводжується проблемою промахів кешу, коли процесор вимушений чекати відповіді від повільної оперативної пам'яті. Унікальна архітектура Zen 3 дозволяє утримувати значні блоки хеш-таблиць або фрагменти гістограм безпосередньо на кристалі процесора. Це кардинально знижує латентність доступу до даних та компенсує обмежену пропускну здатність двоканального контролера

пам'яті DDR4 (51.2 ГБ/с).

Обсяг системної оперативної пам'яті становить 64 гігабайти. В контексті аналізу великих масивів даних логів це є суттєвим апаратним обмеженням. Спроба матеріалізації повного датасету, розгортання послідовностей подій або генерація комбінацій ознак за допомогою стандартних бібліотек неминуче призвела б до повного вичерпання RAM та краху системи. Завдяки імплементації лінивого завантаження та механізмів потокової передачі у Polars, пам'ять використовується виключно як транзитний буфер для обробки мікро-батчів. Таким чином, 64 ГБ є цілком достатніми для ефективної організації процесу обробки, де у пам'яті одночасно утримуються лише оптимізовані індекси та проміжні результати агрегацій.

Хоча етап попередньої підготовки масивів виконується виключно на центральному процесорі, стадія тренування градієнтних ансамблів перекладається на графічний адаптер NVIDIA GeForce RTX 4090. Цей прискорювач має 24 гігабайти надшвидкої відеопам'яті VRAM стандарту GDDR6X та більше 16 тисяч ядер CUDA. Бібліотеки LightGBM та CatBoost мають нативну підтримку виконання обчислень на GPU. Під час пошуку оптимальних вузлів розщеплення у тисячах дерев, математичні операції розпаралелюються на ядрах CUDA, що забезпечує колосальне прискорення процесу навчання у порівнянні з CPU. Обсяг у 24 ГБ VRAM гарантує, що повністю скомпільовані тренувальні датасети з директорії TRAINING_DATASETS_DIR будуть безперервно знаходитися у відеопам'яті без необхідності постійного обміну даними через шину PCIe, що є критичним для забезпечення високої швидкості тренувальних епох.

Для усунення вузьких місць на етапі вводу-виводу в інфраструктурі реалізовано дворівневу архітектуру зберігання даних:

1. NVMe SSD обсягом 1024 ГБ, який є твердотільним накопичувачем, що працює через інтерфейс PCIe 4.0, виступає у якості "гарячого" рівня сховища. Усі проміжні артефакти, збережені ознаки, готові датасети та тимчасові файли при операціях скидання з оперативної пам'яті під час

потоків JOIN операцій записуються сюди. Багатогігабайтна пропускна здатність NVMe дозволяє колонковим оптимізаціям Polars не простоювати в очікуванні надходження байтів з диску.

2. HDD обсягом 8 ТБ це класичний магнітний диск, який використовується для "холодного" зберігання масивних сирих файлів-дампів. За допомогою фреймворку DVC (Data Version Control) забезпечується відстеження версій цих масивних артефактів без їх постійного перебування на швидкісному, але дорогому носії NVMe [28].

Узагальнення технічних специфікацій віддаленого сервера та призначення компонентів у конвеєрі наведені у таблиці 3.1.

Таблиця 3.1 — Технічні специфікації віддаленого сервера та призначення компонентів у конвеєрі

Апаратний компонент	Технічні характеристики	Роль в обчислювальній архітектурі проєкту
Процесор (CPU)	AMD Ryzen 9 5950X (16 ядер / 32 потоки, 64 MB L3 Cache)	Розпаралелення агрегацій у Polars, прискорення хеш-з'єднань за рахунок великого L3-кешу
Оперативна пам'ять (RAM)	64 ГБ DDR4	Кешування транзитних батчів під час лінійних обчислень, уникнення помилок OOM
Графічний адаптер (GPU)	NVIDIA GeForce RTX 4090 (24 ГБ VRAM)	Апаратне прискорення навчання GBDT моделей через CUDA
Швидкісний накопичувач (NVMe)	1024 ГБ SSD (PCIe 4.0)	"Гарячий" рівень для датасетів та Parquet-файлів
Архівний накопичувач (HDD)	8 ТБ HDD	"Холодний" рівень для історичних логів та версіонування DVC

Спроектowana обчислювальна екосистема органічно доповнює обрані

програмні інструменти. Використання лінійних обчислень фреймворку Polars дозволяє обійти обмеження обсягу оперативної пам'яті, використовуючи багатопотокову міць 32 процесорних потоків та пропускну здатність накопичувачів NVMe. У свою чергу, інтеграція алгоритмів GBDT дозволяє перенести найтяжчі ітеративні математичні обчислення на графічний прискорювач, що у комплексі гарантує можливість побудови точних і стійких моделей на основі гігантських просторів поведінкових даних.

Узагальнену архітектуру розробленої системи предиктивної аналітики, що демонструє апаратно-програмний розподіл обчислювального навантаження та інтеграцію баз даних, наведено в додатку Б.

3.2 Розробка конвеєра обробки даних

Системи поведінкової аналітики та прогнозування вимагають побудови масштабованих та відмовостійких архітектур для обробки надвеликих масивів даних. Зростання обсягів згенерованих користувачами даних у мобільних застосунках, таких як геолокаційна інформація та системні логи, набуло експоненційного характеру, що робить традиційні підходи до обробки даних вкрай неефективними. У контексті розробки системи прогнозування конверсії на основі сирих логів мобільного застосунку, архітектура конвеєра Extract, Transform, Load (ETL) відіграє фундаментальну роль, адже вона повинна не лише забезпечувати необхідну пропускну здатність для обробки мільйонів подій щоденно, але й гарантувати абсолютну консистентність даних, багаторівневий захист від витоку цільової змінної та формування складних поведінкових ознак у режимі пакетних обчислень.

Еволюція конвеєрів обробки даних вказує на перехід від монолітних структур до розподілених систем, здатних імплементувати складну логіку трансформації безпосередньо у процесі руху даних. На відміну від класичних

аналітичних завдань, де переважають реляційні структури, обробка потоків мобільних подій стикається з проблемою гетерогенності, високого рівня шуму та порушення хронологічного порядку надходження пакетів даних, спричиненого специфікою мобільних мереж та кешуванням на стороні клієнта. Затримки у передачі даних або їх дублювання можуть суттєво викривити результати предиктивного моделювання. Отже, конвеєр спроектовано таким чином, щоб інтегрувати механізми дедуплікації на основі унікальних ідентифікаторів подій та впорядкування за серверним часом надходження поряд із локальним часом пристрою, що створює надійний фундамент для подальшого вилучення ознак.

Практична реалізація конвеєра спирається на парадигму пакетної обробки, яка є оптимальною для завдань історичного аналізу, навчання моделей машинного навчання та формування складних когорт користувачів, оскільки дозволяє агрегувати повний контекст поведінки за визначені часові вікна перед настанням цільової події. Проектування логіки трансформацій здійснено з огляду на необхідність створення єдиного джерела істини, де кожен етап очистки, фільтрації та екстракції регламентований конфігураційними файлами, що забезпечує відтворюваність експериментів та стійкість до змін у структурі вхідних логів.

Процес побудови ETL-конвеєра розпочинається зі стадії ідентифікації та формалізації вимог до якості вхідних даних та очікуваної структури вихідного ознакового простору. Зважаючи на архітектурні патерни, описані в літературі щодо проектування аналітичних систем для мобільних логів, система повинна розв'язувати завдання парсингу напівструктурованих масивів, зведення сесій користувачів, обчислення часових інтервалів між подіями та математичної трансформації поведінкових патернів у вектори. Усі обчислювальні скрипти, розроблені в межах практичної частини, інкапсулюють ці етапи у послідовність модулів, кожен з яких відповідає за вузькоспеціалізований аспект підготовки даних.

Функціональна архітектура розробленого конвеєра передбачає

наявність трьох модулів. Перший модуль відповідає за первинну фільтрацію, нормалізацію та дедуплікацію сирих логів із забезпеченням контролю ізоляції даних. Другий модуль імплементує бізнес-логіку категоризації подій та розрахунок відносних метрик фрустрації та успішності. Третій модуль зосереджений виключно на трансформації послідовностей дискретних подій у неперервний векторний простір за допомогою алгоритмів обробки природної мови, що дозволяє алгоритмам машинного навчання адекватно сприймати секвенціальну природу людської поведінки у цифровому середовищі.

Функціональну схему роботи розробленого алгоритму, яка відображає послідовність етапів трансформації даних від сирих логів до математичної класифікації користувачів, наведено в додатку В.

3.2.1 Збір, фільтрація та очистка сирих логів

Процес агрегації вхідних даних реалізовано через модуль `HistoricalDataLoader`, який керує двома паралельними потоками завантаження: маркетинговими даними з реляційної бази `MySQL` та подієвою телеметрією зі сховища `Google BigQuery`. Маркетингові дані вилучаються за допомогою комплексних `SQL`-запитів, які об'єднують численні таблиці, зокрема, таблиці підписок та загальних даних користувачів, обчислюють фінансові показники а здійснюють категоризацію рекламних кампаній за допомогою регулярних виразів. У результаті отримується таблиця так званих маркетингових даних формату 1 рядок = 1 користувач, де наведена вся статична інформація. Для запобігання переповненню оперативної пам'яті читання з `MySQL` відбувається частинами по 100 тисяч рядків із подальшим інкрементальним записом у формат `Parquet`. Паралельно, логіка завантаження продуктових подій із `BigQuery` налаштована на ефективну ітеративну обробку в режимі `Arrow-батчів`, зберігаючи їх у партиційовані `Parquet`-файли для забезпечення

ідемпотентності процесу екстракції та уникнення надлишкових запитів. На цьому етапі вважаємо, що всі часові поля приведені до єдиного часового поясу UTC [9].

Після етапу фізичного завантаження, початковий етап функціонування ETL-конвеєра орієнтований на трансформацію сирого масиву телеметричних даних у структурований набір, придатний для обчислення статистичних та поведінкових метрик. Сирі дані, згенеровані мобільними пристроями, характеризуються високим ступенем ентропії: вони містять артефакти тестових запусків, події від пристроїв із застарілими версіями операційних систем, пропуски у життєво важливих полях та аномальні часові мітки. Процес очистки та фільтрації є одним із найважливіших, оскільки якість предиктивної моделі прямо пропорційна якості вхідного сигналу.

Практична реалізація відбору когорт розпочинається з просторової та платформеної ізоляції. Відповідно до методології, описаної в попередніх розділах, дослідження сфокусовано виключно на користувачах операційної системи iOS. Екосистема iOS демонструє значно вищий рівень апаратної та програмної гомогенності порівняно з фрагментованим ринком Android. Це дозволяє нівелювати вплив апаратних затримок на метрики когнітивного навантаження та фрустрації. Таким чином, зафіксована у логах затримка між кліками є наслідком когнітивного процесу користувача, а не технічного лагу пристрою. Також політики конфіденційності Apple, зокрема впровадження фреймворку App Tracking Transparency (ATT), створюють унікальні умови для атрибуції та формування ідентифікаторів сесій, які вимагають специфічного підходу до зшивання івентів. У програмному коді ця фільтрація реалізована через правило парсингу метаданих подій: рядки, де значення поля операційної системи не відповідає патернам iOS або версії платформи нижче визначеного мінімуму, повністю вилучаються з подальшого конвеєра обробки.

Наступним кроком є обробка пропусків у даних. Стратегія роботи з пропусками диференційована та ґрунтується на природі виникнення відсутніх даних: чи є вони випадковими, залежними від інших змінних або системними.

Для категоріальних атрибутів пристрою або сесії, наприклад, мова локалізації чи країна, скрипт не застосовує видалення рядків, оскільки це призвело б до втрати цінного поведінкового контексту. Натомість такі пропуски замінюються на виділену константу `Unknown`. У контексті предиктивного моделювання сам факт відсутності певного системного атрибута може виступати потужним предиктором; наприклад, неможливість застосунку визначити геолокацію може корелювати з відмовою користувача надати відповідні дозволи, що, у свою чергу, корелює зі зниженою довірою до продукту і, відповідно, нижчою ймовірністю оформлення підписки. Для числових значень, таких як тривалість відображення екрану, у випадку аномалій або від'ємних значень, що виникають при розсинхронізації внутрішнього годинника пристрою, алгоритм застосовує медіанну імпутацію в межах сесії або примусове обнулення, спираючись на конфігураційні правила конвеєра.

Фундаментальною складовою даного підрозділу є імплементація превентивних механізмів захисту від витіку цільової змінної. У машинному навчанні витік даних визначається як ситуація, коли модель під час навчання отримує доступ до інформації, яка фізично або логічно не може бути доступною в момент здійснення прогнозу у реальних умовах. У задачах прогнозування конверсії, використання логів поведінки користувача, що відбулися одночасно з цільовою подією або після неї, призводить до негативних наслідків: модель демонструватиме ідеальні метрики точності на етапі крос-валідації, оскільки вона фактично "підглядає" у майбутнє, але виявиться абсолютно недієздатною на нових даних. Як зазначено в дослідженнях, моделі, навчені на таких контамінованих масивах даних, не здатні до генералізації, оскільки вони вивчають патерни, що є наслідком цільової дії, а не її першопричиною.

Для радикального викорінення проблеми `target leakage` у розробленому коді застосовано методологію суворого хронологічного усічення подій, що є стандартом де-факто у надійних предиктивних системах. Логіка скрипта

реалізована через послідовність операцій з датафреймом. На першому етапі скрипт ідентифікує цільову подію для кожного унікального користувача. У контексті даної дипломної роботи цільовою подією є момент успішного оформлення підписки, а саме змінна `is_converted`, яка набуває значення 1. Скрипт створює референсну таблицю, що складається з двох колонок: `user_id` та `target_timestamp` — точний час настання цільової події. На другому етапі ця референсна таблиця приєднується до загального масиву сирих логів. На третьому етапі застосовується фільтр: алгоритм порівнює часову мітку кожної окремої події t_{event} з часовою міткою цільової події t_{fixed} для відповідного користувача. Усі події, які задовольняють умову $t_{event} \geq t_{fixed}$, видаляються з навчальної та тестової вибірок. Застосування такого протоколу усічення гарантує, що ознаковий простір формується виключно на основі тих поведінкових сигналів, які передували прийняттю рішення про покупку. Це імітує реальні умови роботи моделі в інфраструктурі застосунку, де система повинна оцінити ймовірність підписки на основі поточного неповного досвіду користувача.

Крім прямого витоку цільової змінної, розроблений конвеєр мінімізує ризики виникнення часткового витоку через ознаки, або *feature leakage*. У процесі агрегації скрипт слідкує за тим, щоб жодна з похідних ознак не містила неявної інформації про майбутнє. Наприклад, агреговані змінні на кшталт "загальна кількість днів у застосунку" або "життєва цінність користувача" обчислюються динамічно станом на момент кожної конкретної сесії до t_{fixed} .

Після завершення етапів фільтрації та очистки, скрипт ініціює масштабний процес генерації ознак. Налаштування конфігураційного файлу дозволяють гнучко змінювати параметри агрегації, такі як розмір ковзних вікон, наприклад, активність за останні 24 години, 3 дні, 7 днів, пороги визначення нових сесій та мінімальну кількість подій для валідної сесії. Для забезпечення наочності та структурованості репрезентації логіки коду, усі згенеровані ознаки об'єднані у чотири магістральні категорії. Детальний опис цих типів ознак наведено у таблиці 3.2.

Таблиця 3.2 — Структура та механізми обчислення ознакового простору поведінкової аналітики

Категорія ознак	Опис та семантичне значення	Приклади змінних	Алгоритм обчислення та логіка скрипта
Статичні	Незмінні атрибути пристрою та інсталяції.	country, media_source, app_version	Вилучаються з метаданих першого логу за user_id. Рідкісні значення групуються в «Other» за заданим.
Часові	Розподіл активності в часі, частота сесій та загальна тривалість залученості.	ctx_install_to_trial_sec, ratio_pretrial_duration_sec, trial_cos_visit_search	Обчислюються віконними функціями як різниця між сусідніми часовими мітками.
Поведінкові	Кількісні метрики інтенсивності взаємодії до моменту усічення.	trial_cnt_sent_positive, trial_cnt_plant_add, trial_cnt_sent_push	Групування за user_id та session_id. Підрахунок частот подій, середніх і максимальних значень.
Контекстні	Фіксують взаємодію зі специфічними вузлами застосунку.	trial_cnt_plant_added, trial_rate_onboarding_pos, pretrial_rate_push_pos	Фільтрація подій за таксономією. Генерація бінарних прапорців відвідування та кумулятивних сум взаємодій з модулями.

Така вичерпна декомпозиція сирих логів на структуровані вектори ознак дозволяє захопити як інтенсивність, так і контекстуальну спрямованість поведінки. Проте, звичайні агрегації не здатні повною мірою відобразити якість користувацького досвіду. Для розв'язання цього завдання розроблено модуль конвеєра, присвячений розрахунку метрик фрустрації.

3.2.2 Логіка генерації статичних та часових ознак

У розробленому конвеєрі обробки даних за формування статичних, демографічних та фінансових атрибутів користувача відповідає спеціалізований клас `StaticTransformer`. Цей модуль спроектовано для роботи з незмінними характеристиками профілю такими як країна, тип пристрою, джерело залучення тощо. Логіка роботи класу алгоритмічно розділена на дві фази: агрегацію когортних статистик, безпосередню трансформацію векторів на рівні окремого користувача.

На етапі агрегації та підготовки словників скрипт формує необхідні артефакти на основі історичних даних за останні 30 днів, відштовхуючись від дати оформлення тріалу. Це дозволяє динамічно адаптуватися до змін у розподілі користувачів. Усі числові колонки, визначені у списку `config.num_cols`, наприклад, `spend_usd`, `price`, `cpv`, `cpm`, `trial_period`, примусово приводяться до типу `Float64`, що забезпечує математичну консистентність під час подальшого розрахунку середніх значень. Для боротьби з великими розмірностями алгоритм аналізує категоріальні змінні з `config.cat_cols` такі як `country`, `device_type`, `source`, скрипт ранжує їх за сумарним показником витрат або за частотою виникнення серед користувачів. У словнику `top_n_maps` зберігаються лише N найпопулярніших значень згідно з `config.top_n_config`, наприклад, топ-10 для `country`, топ-5 для `device_type` та `source`. Скрипт також підраховує загальну кількість входжень кожного унікального значення для

категоріальних ознак. Ці дані зберігаються у словниках `freq_maps` для подальшого частотного кодування, після чого датафрейм групується за набором колонок `cohort_cols`, комбінація дати покупки, країни та джерела. Для кожної такої когорти обчислюються середні значення для всіх числових метрик. У результаті, ця агрегована таблиця стає еталоном для порівняння поведінки індивідуального користувача з його когортою.

Для трансформації ознак на рівні користувача, метод `transform` приймає масив даних та застосовує раніше обчислені артефакти для формування фінального ознакового простору кожного користувача. Щоб гарантувати статичність вектора та відсутність дублів, скрипт виконує групування за `user_id` і залишає лише перший хронологічний запис для кожного користувача. Застосовується логіка заповнення відсутніх даних, пропуски у числових колонках заповнюються нулями, а у категоріальних — константою "unknown". Усі категоріальні значення користувача перевіряються за словником `top_n_maps`. Якщо, наприклад, країна користувача не входить до топ-10 найпопулярніших, її значення примусово замінюється на рядкову заглушку "Other", що унеможлиблює перенавчання моделі на поодиноких і нетипових прикладах. Скрипт автоматично створює нові нелінійні ознаки шляхом конкатенації визначених у `config.interaction_pairs` пар колонок, наприклад, поєднання `country` та `device_type`, або `device_type` та `is_big_city`. Згенеровані колонки отримують назви у форматі `inter_{col_a}_{col_b}`, це дозволяє моделі вловлювати складні патерни, наприклад, специфічну поведінку користувачів планшетів з певного регіону. Текстові категоріальні значення замінюються на їхні числові частоти зі словника `freq_maps`. Якщо у тестовій вибірці зустрічається категорія, якої не було під час розрахунку статистики, скрипт присвоює їй частоту 1 за замовчуванням. Профіль користувача з'єднується з таблицею когортних статистик `lf_stats_rolling`, прив'язка відбувається за хронологічним зсувом `key_date_pretrial`, визначеним у `config.pre_trial_recent_hours`. Далі для кожної числової метрики розраховується відносний показник за принципом ділення значення показника

для фіксованого користувача на середнє значення по його когорті, де до знаменника з метою уникнення ділення на нуль додається 0.0001. Ці змінні, з префіксом `ratio_{prefix}_{metric}`, дозволяють алгоритму машинного навчання кількісно оцінити індивідуальні відхилення: наприклад, зрозуміти, чи витрачає користувач удвічі більше часу в застосунку порівняно із середнім статистичним представником його ж когорти. На фінальному етапі всі категоріальні колонки та ознаки взаємодії приводяться до системного типу `Categorical`.

За захоплення хронологічної семантики, розрахунок сесійних інтервалів та оцінку швидкості взаємодії користувача із застосунком відповідає спеціалізований модуль `TimeFeaturesTransformer`. На відміну від статичних атрибутів, часові ряди поведінкових логів вимагають складніших математичних трансформацій для приведення неструктурованих часових міток до формату, придатного для алгоритмів машинного навчання. При цьому, програмна логіка класу алгоритмічно розділена на аналогічні до статичних ознак етапи.

Для кожної події скрипт обчислює різницю у секундах між часом настання події та цільовою часовою міткою початку тріалу. Оскільки активність користувачів має виражений добовий ритм, використання звичайної години доби від 0 до 23 як лінійної змінної є математично некоректним, оскільки 23:00 і 01:00 є близькими в реальності, але далекими лінійно. Для розв'язання цієї проблеми скрипт застосовує тригонометричні трансформації: годину події, нормалізовану на 24 години, переведено у синус та косинус, що дозволяє моделі розпізнавати циклічні патерни поведінки. Увесь масив подій розділяється на аналітичні вікна за допомогою фільтрів, а саме виконані користувачами події в діапазоні від мінус `pre_trial_recent_hours` до 0 секунд відносно моменту початку тріалу та від початку тріалу до `trial_window_hours` годин після, де `trial_window_hours` менше ніж загальна тривалість пробного періоду. Також алгоритм ітерує по списку подій `config.target_events`, де для кожної такої події обчислюється її абсолютна

частота, середнє значення циклічного часу та мінімальний час від початку аналізованого вікна до першого прояву цієї конкретної дії. Ще однією важливою метрикою є ентропія Шеннона, яка кількісно оцінює різноманітність виконаних дій, обчислюючи ентропію розподілу унікальних подій у межах вікна, нормалізовану до шкали від 0 до 1. Висока ентропія свідчить про "блукання" інтерфейсом, що корелює з когнітивним навантаженням. Для кожного часового вікна скрипт генерує вектор показників:

- 1) `count` — сумарна кількість івентів у вікні;
- 2) `duration_sec` — тривалість сесії у секундах, різниця між останньою та першою подією;
- 3) `velocity` — швидкість взаємодії, що розраховується як відношення кількості подій до тривалості;
- 4) `gap_mean` — середній час затримки між двома послідовними діями.

Для того, щоб поведінка користувача не розглядалася у вакуумі, скрипт обчислює еталонні показники на глобальному рівні. Спочатку викликається метод базових обчислень для всієї вибірки. Далі датафрейм групується за ідентифікаторами когорти: датою початку тріалу, країною географічного розташування та джерелом трафіку. Для кожної числової метрики з префіксами вікон `pretrial_` або `trial_` обчислюється математичне середнє, а нові колонки отримують префікс `avg_`.

Фінальними етапами розрахунку часових ознак слугують аналогічні до статичних обчислення відношення індивідуального показника до когортного та очищення даних від пропусків, замінюючи їх нулями [20].

3.2.3 Програмна реалізація багаторівневої таксономії та метрик фрустрації

Сирі назви подій, які надходять від мобільних клієнтів, здебільшого є вкрай неструктурованими та залежать від конкретних рішень розробників на етапі імплементації аналітики. Використання таких сирих значень безпосередньо як вхідних даних для алгоритмів машинного навчання є хибною практикою: вона призводить до надмірної розрідженості простору ознак, виникнення дублікатів з різними назвами та повної втрати семантичного зв'язку між концептуально спорідненими діями. Щоб подолати цю проблему, в коді розробленого конвеєра імплементовано програмну абстракцію у вигляді багаторівневої таксономії подій.

Багаторівнева таксономія є системою класифікації, яка виконує семантичний мапінг кожної сирої події на ієрархічну структуру, що складається з кількох уніфікованих рівнів абстракції. Згідно з логікою скрипта, кожен лог проходить через систему регулярних виразів та словників, що дозволяє звести тисячі унікальних назв до контрольованого словника категорій. Архітектура таксономії, імплементована у скрипті, включає рівні категорії та дії. Категорія специфікує функціональний контекст всередині домену, наприклад: `search`, `push`, `onboarding`, `botanist`. Дія описує фізичну або логічну взаємодію користувача з інтерфейсом, наприклад, `plant_added`, `search_initiated`, `image_open`, `video_play`.

Цей етап трансформації є фундаментальним, оскільки він дозволяє агрегувати поведінку на різних рівнях деталізації. Модель машинного навчання здатна аналізувати патерни як на мікрорівні, так і на макрорівні. Що більш важливо, стандартизована таксономія є обов'язковою передумовою для алгоритмічного виявлення та розрахунку метрик фрустрації та когнітивного зусилля користувача.

Серед ключових показників якості користувацького досвіду,

реалізованих у коді, виділяється коефіцієнт успішності виконання завдань, або Task Success Rate (TSR). TSR є однією з найважливіших метрик поведінкової аналітики, оскільки вона кількісно оцінює ефективність та інтуїтивність взаємодії з продуктом. У контексті комерційних застосунків високий показник TSR для базових мікро-завдань корелює зі зниженням тертя в інтерфейсі, що, як наслідок, підвищує ймовірність конверсії. Низький TSR, навпаки, є безпосереднім сигналом про наявність бар'єрів, незрозумілого дизайну або технічних помилок, що спонукають користувача до відтоку.

Математична основа TSR є інтуїтивно зрозумілою: це бінарна метрика на рівні окремої спроби, яка агрегується у відсоткове відношення і описується формулою (2.3).

Програмна імплементація розрахунку TSR вимагає значно складнішої логіки, ніж проста агрегація. Скрипт повинен вміти ідентифікувати у потоці логів події, які слугують тригерами початку завдання, та події, які підтверджують його успішне завершення. У коді це реалізовано через механізм сесійних вікон та аналіз воронок.

Для кожного користувача агрегується загальний вектор `user_tsr` по всіх ключових воронках. Аналіз довірчих інтервалів для TSR показує, що ця метрика є стабільним предиктором: користувачі з високим TSR демонструють впевненість у навігації, що підвищує їхню схильність до дослідження платних функцій.

Однак TSR вимірює лише результативність, залишаючи поза увагою "ціну", яку користувач заплатив за досягнення цього результату. Для квантифікації цієї "ціни" у конвеєрі імплементовано розрахунок Cognitive Effort Score (CES) — оцінки когнітивного зусилля. У когнітивній психології та науках про поведінку розумове зусилля формалізується через теорію активного висновування, де зусилля розглядається як відхилення від попередніх переконань про оптимальну дію, тобто необхідність долати ментальні бар'єри та адаптуватися до неочікуваних або складних інтерфейсів. У контексті аналізу системних логів ми не маємо доступу до фізіологічних

показників, наприклад, руху очей чи серцевого ритму, тому когнітивне навантаження повинно бути виведене алгоритмічно через евристичні проксі-метрики.

У розробленому скрипті CES операціоналізується як математичне відношення загального обсягу виконаних дій до їхньої унікальності. Фундаментальна гіпотеза полягає в тому, що впевнений користувач, який не відчуває когнітивного перевантаження, рухається інтерфейсом цілеспрямовано. Його шлях характеризується лінійністю та високим відсотком унікальних кроків $A \rightarrow B \rightarrow C \rightarrow D$. Натомість користувач, який зіткнувся з фрустрацією, заплутаним навігаційним меню або неочікуваною поведінкою системи, починає діяти хаотично. Ця "загубленість" проявляється у циклічних поверненнях на попередні екрани, повторних кліках по неактивних елементах та багаторазових спробах виконати одну й ту саму дію. У такому сценарії шлях стає надлишковим, наприклад $A \rightarrow B \rightarrow A \rightarrow B \rightarrow C \rightarrow C \rightarrow C \rightarrow D$. Спираючись на цей підхід, скрипт обчислює CES за формулою (2.4).

Що вище значення CES, то більше повторюваних та потенційно непотрібних дій здійснив користувач, що свідчить про високий рівень тертя та надмірне когнітивне навантаження.

Для поглиблення аналізу фрустрації, окрім простого Ratio, у коді реалізовано розрахунок більш складної інформаційно-теоретичної метрики — ентропії дій. Ентропія дій базується на формулі Шеннона і вимірює ступінь непередбачуваності або хаотичності послідовності подій користувача. Логіка скрипта передбачає обчислення крос-ентропії або безпосередньої ентропії розподілу частот різних категорій дій у сесії за формулою (2.5).

Дослідження підтверджують, що сесії з аномально високою ентропією дій є надійним індикатором сценаріїв з високим когнітивним зусиллям, таких як постійне перемикання уваги між екранами або намагання розібратися зі складним функціоналом. Інтеграція TSR, CES та Action Entropy до загального вектора ознак дозволяє системі не просто фіксувати, що зробив користувач,

але й розуміти, як саме він це зробив, ідентифікуючи приховані предиктори відмови від підписки [3].

3.2.4 Векторизація поведінкових послідовностей

Попри високу інформативність статистичних агрегацій та обчислених метрик фрустрації, вони володіють одним фундаментальним недоліком: повним ігноруванням секвенціальної природи людської поведінки. Статичні ознаки здатні зафіксувати, що користувач відвідав екран оплати 5 разів і зіткнувся з помилкою 2 рази, але вони не зберігають інформацію про те, у якому порядку ці події відбувалися. Проте саме впорядкованість несе найглибший семантичний зміст. Для захоплення цієї прихованої хронологічної семантики у третьому модулі конвеєра застосовано потужні інструменти обробки природної мови (Natural Language Processing, NLP), де історія дій розглядається як своєрідний "текстовий документ", а окремі дискретні події, закодовані за допомогою розробленої таксономії, виступають у ролі "слів".

У розробленій парадигмі формується матриця, де фіксується лише наявність або частота появи кожного слова в документі. Проте, як зазначено вище, такий підхід повністю руйнує контекст і не розрізняє порядок "слів". У поведінковій аналітиці, так само як і в лінгвістиці, перестановка елементів радикально змінює сенс. Тому скрипт векторизації відмовляється від примітивних уніграм на користь генерації n -грамів.

N -грами являють собою безперервні послідовності з n сусідніх елементів у масиві. Замість того, щоб аналізувати події ізольовано, скрипт ітерує по хронологічно відсортованій історії користувача та формує послідовності довжини 2 (біграми) та 3 (триграми). Параметр генерації охоплює діапазон від уніграм до триграм (`ngram_range=(1,3)`), що дозволяє

моделі аналізувати як базові частоти подій, так і складні транзакційні переходи. У такий спосіб алгоритм автоматично конструює тисячі потенційно значущих патернів без необхідності ручного втручання аналітиків для формулювання правил.

Однак застосування n -грамів призводить до виникнення математичної та обчислювальної проблеми, відомої як "прокляття розмірності". Кількість можливих комбінацій подій зростає експоненційно зі збільшенням n , перетворюючи ознаковий простір на надзвичайно широку та розріджену матрицю. Більше того, просте розраховування частот генерує зміщені оцінки: найбільш розповсюджені рутинні події будуть мати найбільші математичні ваги у векторах усіх користувачів. Проте саме ці масові події несуть найменше дискримінативної інформації — вони притаманні як користувачам, які куплять підписку, так і тим, хто ніколи цього не зробить. Значення мають специфічні, можливо рідкісніші, але контекстуально важливі патерни.

Для вирішення цієї проблеми у скрипті імплементовано алгоритм зважування Term Frequency-Inverse Document Frequency (TF-IDF). Це класичний метод інформаційного пошуку, який елегантно балансує вагу кожної ознаки. Він винагороджує n -грами, які часто зустрічаються в історії конкретного користувача, але суворо штрафує ті n -грами, які повсюдно присутні в логах усіх інших користувачів.

Програмна реалізація цього математичного апарату виконана з акцентом на продуктивність та ефективність використання оперативної пам'яті. У коді конвеєра скрипт векторизації функціонує за наступним алгоритмом.

На першому кроці скрипт агрегує масив логів: для кожного унікального `user_id` усі його події, які залишилися після обов'язкового етапу усічення, сортуються у хронологічному порядку. Кожна подія трансформується у свій текстовий еквівалент згідно з таксономією, наприклад, використовуючи рівень Action або об'єднаний Label. Ці події з'єднуються у єдиний довгий рядок, розділений пробілами, що фізично імітує текстовий документ.

На другому кроці ініціалізується клас векторизатора, що є аналогом

TfidfVectorizer з бібліотеки Scikit-learn у Python або відповідного трансформера у Spark MLlib. Векторизатор здійснює токенизацію стрічок та формує n -грами в діапазоні від 1 до 3.

Третім кроком є редукція розмірності ознакового простору та фільтрація шумів за формулою (2.7). У скрипті задано гіперпараметр мінімальної частоти документа, наприклад, `min_df=0.01` — 1% від розміру когорти або фіксоване значення `min_df=5`. Цей механізм вказує векторизатору ігнорувати будь-які n -грами, які зустрічаються рідше, ніж у заданому порозі користувачів. Це логічне та потужне рішення: унікальний баг, з яким зіткнувся лише один користувач, або вкрай нетипова випадкова послідовність кліків генерує унікальну триграму. Якщо залишити такі артефакти у навчальній вибірці, алгоритм машинного навчання швидко перенавчиться на цих шумах, асоціюючи їх із цільовою змінною. Фільтрація через `min_df` відкидає статистичний шум, залишаючи лише ті поведінкові патерни, які мають репрезентативне підтвердження у даних. Окрім того, це на порядки зменшує кількість стовпців у фінальній матриці.

На фінальному, четвертому кроці, скрипт викликає метод `fit_transform` на навчальній вибірці для обчислення словника і ваг IDF, та метод `transform` для тестової вибірки. Результатом роботи скрипта є формування розрідженої матриці. У цій матриці рядки є векторами користувачів, стовпці відповідають унікальним поведінковим n -грамам, а комірки містять дійсні числа — обчислені показники TF-IDF за формулою (2.6). Завдяки використанню CSR-формату, у пам'яті фізично зберігаються лише ненульові значення, що дозволяє конвеєру ефективно утримувати ознаковий простір десятків тисяч користувачів на стандартному обладнанні [11, 29].

3.3 Відбір ознак та формування фінального високорозмірного датасету

Формування фінального датасету для задач предиктивної аналітики та прогнозування конверсії користувачів мобільного застосунку є фундаментальним інженерним та математичним етапом, що інтегрує результати ізольованих процесів екстракції ознак у єдину узгоджену структуру. Зважаючи на гетерогенну природу вхідних сигналів, які охоплюють весь спектр користувацької активності — від статичних демографічних атрибутів до динамічних метрик фрустрації та розріджених векторів обробки природної мови — класичні підходи до конкатенації даних виявляються алгоритмічно та апаратно неефективними. Процес об'єднання вимагає застосування спеціалізованих парадигм інтеграції модальностей, суворих математичних трансформацій для стабілізації дисперсії простору ознак та передових алгоритмів редукції розмірності для подолання фундаментальної проблеми машинного навчання, відомої як "прокляття розмірності".

Даний розділ детально висвітлює архітектурні, математичні та програмні рішення, імплементовані в розробленому конвеєрі обробки даних, що забезпечили коректне злиття щільних та розріджених просторів ознак із подальшою оптимізацією сформованого простору для алгоритмів градієнтного бустингу над деревами рішень. Успішна імплементация цього етапу є базовою передумовою для забезпечення стабільної роботи моделей в умовах апаратних обмежень, описаних у попередніх розділах.

3.3.1 Теоретичне та архітектурне обґрунтування парадигми "раннього злиття"

Інтеграція даних різної модальності та природи в сучасних системах машинного навчання традиційно реалізується за допомогою однієї з трьох магістральних парадигм: раннього злиття, пізнього злиття або проміжних гібридних підходів. У контексті даного дипломного дослідження, де стратегічною метою є передбачення ймовірності настання цільової події *is_converted* на основі комплексного синтезу демографії, фінансових сигналів, часових метрик фрустрації та TF-IDF векторів поведінкових секвенцій, базовою архітектурною моделлю було обрано стратегію раннього злиття на рівні ознак.

Парадигма раннього злиття функціонує за принципом конкатенації різнорідних просторів ознак на етапі, що безпосередньо передуює процесу навчання моделі. Математично цей процес описується як створення розширеної матриці X_{fused} , де стовпці формуються шляхом горизонтального об'єднання векторів щільних поведінкових метрик X_{dense} та розріджених текстових репрезентацій X_{sparse} . На відміну від стратегії пізнього злиття, де для кожної ізольованої модальності, наприклад, окремо для текстових TF-IDF n-грамів та окремо для демографічних профілів, будується незалежна базова предиктивна модель, а фінальний результат формується шляхом ансамблювання їхніх ймовірнісних прогнозів, раннє злиття дозволяє єдиному центральному алгоритму машинного навчання самостійно та безпосередньо виявляти крос-модальні взаємозв'язки.

У сфері поведінкової аналітики мобільних застосунків ізольований аналіз окремих модальностей неминуче призводить до втрати контекстуальної цілісності. Наприклад, наявність специфічного патерну в розрідженому TF-IDF векторі, такого як триграма, може мати кардинально різну предиктивну силу та семантичне навантаження залежно від статичної країни походження

користувача або його джерела залучення. Якщо використовувати пізні злиття, алгоритми не зможуть побудувати спільний простір рішень, оскільки інформація про країну та інформація про помилку платежу оброблятимуться у вакуумі ізольованих моделей. Сучасні алгоритми класу GBDT здатні будувати складні дерева рішень, вузли яких на різних рівнях глибини по чергово розщеплюються за щільними метриками когнітивного навантаження та бінарними компонентами розріджених векторів обробки природної мови, бездоганно захоплюючи складні нелінійні взаємозв'язки між цими паралельними світами користувацького досвіду.

Реалізація стратегії Feature-Level Fusion у розробленому пайплайні ускладнюється безпрецедентними структурними відмінностями модальностей, які підлягають конкатенації. Демографічні та статичні ознаки характеризуються низькою розмірністю, високою інформаційною щільністю та здебільшого категоріальною або цілочисельною природою. Поведінкові часові метрики та індикатори фрустрації представляють собою неперервні дійсні числа, яким притаманні асиметричні розподіли та колосальна дисперсія, наприклад, тривалість активних сесій у мілісекундах або Task Success Rate у відсотках. Натомість, TF-IDF вектори формують високорозмірний простір, що складається з тисяч незалежних колонок, які відображають абсолютні або відносні частоти виникнення уніграм, біграм та триграм. Ці текстові вектори є екстремально розрідженими і оперують значеннями у строго нормалізованому діапазоні від 0 до 1. Для коректного імплементації такого злиття без ризику деградації сигналу, пайплайн застосовує складну послідовність математичних трансформацій, які зводять розрізнені дані до уніфікованого вигляду, зберігаючи при цьому дискримінативну силу кожної підсистеми.

3.3.2 Управління оперативною пам'яттю під час конкатенації розріджених та щільних матричних структур

Процес матеріалізації концепції раннього злиття у програмному кодї мовою Python нашо́вхується на обмеження фізичного апаратного забезпечення. Згідно зі специфікацією, визначеною у попередніх розділах, серверна інфраструктура оперує пулом у 64 ГБ оперативної пам'яті. Традиційне використання популярних бібліотек машинного навчання, таких як Scikit-learn, генерує для TF-IDF ознак стиснуті розріджені матриці формату CSR, які зберігають у пам'яті лише ненульові значення та їхні індекси, ігноруючи нулі. Цей формат є стандартом де-факто для роботи з текстовими ознаками, оскільки він економить гігабайти оперативної пам'яті. Водночас, демографічні ознаки, часові проміжки та складні метрики фрустрації зберігаються у вигляді традиційних щільних структур усередині фреймворку Polars DataFrame [30].

Спроба прямої алгоритмічної конвертації багатогігабайтної CSR-матриці TF-IDF, яка представляє історію дій для сотень тисяч користувачів у тисячовимірному просторі, у щільний DataFrame з метою виконання тривіальної реляційної операції JOIN за первинним ключем `user_id` гарантовано призводить до небажаних наслідків. Конвертація розрідженого формату у щільний означає примусове розміщення мільярдів нулів у комірках оперативної пам'яті, що провокує експоненційне зростання споживання RAM та неминуче аварійне завершення процесу через вичерпання ресурсів.

Для подолання цього фундаментального інженерного бар'єра у програмному модулі `build_historical_dataset_baseline.py` застосовано сучасну парадигму лінивих обчислень на базі рушія Polars. Логіка безпечної конкатенації розріджених та щільних даних реалізується за допомогою ітеративного алгоритму потокової обробки. Замість матеріалізації всього масиву ознак в оперативній пам'яті, кожен клас-трансформер викликається

послідовно. На вхід трансформера подається віртуальний граф виконання запиту, а на виході повертається збагачений проміжний логічний план з унікальними локальними ознаками.

Безпосереднє злиття модальностей відбувається шляхом покрокового приєднання нових стовпців до компактного базового ідентифікаційного фрейма, який первинно містить лише мінімальний набір системних колонок: `user_id`, `original_purchase_date` та цільову змінну конверсії `is_converted`. Polars виконує алгоритм лівостороннього хеш-з'єднання у паралельному багатопотоковому режимі, обмежуючи максимальну кількість одночасних потоків системними змінними середовища, що запобігає конкуренції за кеш-пам'ять процесора на рівні L3.

У процесі виконання численних послідовних операцій `join` між десятками проміжних вітрин даних закономірно виникає проблема дублювання метаданих та колонок-ідентифікаторів. Логіка функції `merge_and_finalize_datasets` у згаданому скрипті містить превентивний автоматизований механізм очищення артефактів злиття: усі похідні стовпці, що містять суфікс `_right`, який Polars автоматично додає до дубльованих колонок із правого датафрейма при конфлікті імен, ідентифікуються за допомогою регулярних виразів та відкидаються. Цей крок не лише звільняє транзитні буфери пам'яті, але й унеможливує виникнення абсолютної мультиколінеарності у фінальній матриці.

Ключовим стабілізаційним елементом у конвеєрі злиття є функція `enforce_schema`, яка відповідає за синхронізацію типів даних та заповнення пропусків. Унаслідок специфіки лівостороннього з'єднання, користувачі, які ніколи не взаємодіяли з певним модулем застосунку, наприклад, не переглядали воронку онбордингу, отримують Null-значення у відповідних колонках. Функція `enforce_schema` ітеративно перевіряє кожну колонку і заповнює такі несистемні пропуски обчислювальними нулями: 0.0 для типів з плаваючою комою, 0 для цілих чисел, або категоріальними заглушками "unknown". Найважливішим аспектом цієї функції є агресивне та примусове

переведення масивних числових векторів формату Float64 до Float32. Ця операція математично зменшує обсяг споживаної оперативної пам'яті для щільних числових блоків рівно вдвічі, не призводячи до втрати точності, оскільки алгоритми дерев рішень оперують грубими гістограмними наближеннями, для яких мантиса 32-бітного числа з плаваючою комою є більш ніж достатньою. Крім того, перед фіналізацією злиття, скрипт застосовує предикативний темпоральний фільтр, що забезпечує хронологічну валідність даних, який гарантує відсутність аномалій, пов'язаних із часовими мітками.

Завдяки архітектурі потокового запису безпосередньо на накопичувач формату Parquet, оптимізатор запитів фрагментує обчислення та записує фінальний результат мікро-батчами, уникаючи матеріалізації конкатенованої матриці X_{fused} у фізичній оперативній пам'яті.

3.3.3 Математична трансформація та нормалізація числового простору

Безпосередньо після об'єднання гетерогенних просторів ознак конвеєр стикається з проблемою масштабування та стабілізації дисперсії. Поведінкові метрики користувачів мобільних застосунків, зокрема, кількість згенерованих помилок сесії, загальний час перебування у специфічних функціональних модулях, демонструють екстремально асиметричні розподіли з важкими "хвостами". У поведінкових логах статистично закономірно зустрічаються аномальні користувачі, чия інтенсивність активності на порядки перевищує середні показники по когорті. Наявність таких незбалансованих викидів здатна суттєво дестабілізувати процес навчання алгоритмів класу GBDT. Хоча дерева рішень вважаються стійкими до монотонних перетворень та масштабування, алгоритми жадібного пошуку оптимальних точок

розщеплення будуть витрачати надмірну кількість обчислювальних ітерацій на ізоляцію цих екстремумів в окремі листки, що призведе до розростання глибини дерев, деградації їхньої узагальнюючої здатності та перенавчання.

Для стабілізації гіпервимірному числового простору у модулі формування фінального датасету реалізовано детермінований двоступеневий процес просторової нормалізації: вінзоризація через відсікання аномалій та зсунута логарифмічна трансформація.

Програмна логіка конвеєра використовує рефлексію схем даних для динамічної ідентифікації масиву числових ознак, відфільтровуючи типи Float32, Float64, та сімейство цілочисельних Int8–Int64, і розраховує для кожної ідентифікованої колонки поріг 99-го процентиля на основі емпіричної кумулятивної функції розподілу тренувальної вибірки. Математично операція відсікання екстремумів для кожного вектора X описується формулою (2.8).

Ця процедура виконує функцію нелінійного просторового стиснення. Її головною предиктивною метою є гарантувати, що аномальна активність не буде спотворювати градієнти функції втрат під час оптимізації. Водночас, імплементація P99 Clipping є значно безпечнішим підходом, ніж абсолютне видалення рядків-екстремумів із датасету. Користувачі з гіперактивною поведінкою фізично залишаються у навчальній та тестовій вибірках, але інтенсивність їхніх числових сигналів обмежується максимальним репрезентативним порогом, дозволяючи моделі навчатися на факті їхньої високої активності без ризику градієнтного вибуху.

Після стабілізації екстремумів до усього пулу підготовлених числових змінних застосовується операція зсунутої логарифмічної трансформації, відома як Log1p за формулою (2.9).

Застосування класичного натурального логарифма $\ln x$ до масиву агрегованих поведінкових логів та NLP-векторів є математично некоректним та унеможливорює подальші обчислення, оскільки простір ознак містить гігантську кількість істинних нулів, наприклад, кількість відвідувань екрану FAQ або кількість викликів модуля watering_calculator для абсолютної

більшості користувачів дорівнює 0, а функція $\ln 0$ не визначена у полі дійсних чисел і продукує помилку NaN. Зсув аргументу на одиницю вирішує цю алгебраїчну перешкоду: нульові значення залишаються незмінними нулями ($\ln(1 + 0) = \ln 1 = 0$), тоді як усі додатні числа компресуються вздовж логарифмічної кривої.

Ця трансформація здійснює важливий вплив на здатність моделі машинного навчання узагальнювати дані. Переводячи приховані експоненційні залежності у лінійну площину та симетризуючи скошені вправо розподіли, Log1p дозволяє алгоритмам значно краще вловлювати маргінальну корисність кожної додаткової дії користувача. З погляду когнітивного навантаження та поведінкової психології, різниця між відсутністю помилок та п'ятьма помилками має колосальний негативний вплив на вірогідність конверсії, тоді як різниця між 100 та 105 помилками є семантично незначущою. Трансформація Log1p відображає цю поведінкову метрику у стиснутому числовому просторі, допомагаючи гістограмним оптимізаторам моделей машинного навчання знаходити більш оптимальні пороги дискретизації.

Особливістю реалізації пайплайна є селективність процесу нормалізації. Обчислювальний скрипт містить предикативну логіку, яка захищає певні класи змінних від руйнівного впливу Clipping та Log1p-трансформацій.

Ключові змінні, такі як `user_id`, часова мітка `original_purchase_date`, ідентифікатор країни `country`, медіа-джерело та головна цільова змінна `is_converted`, занесені до конфігураційних списків `EXCLUDE_FROM_CLEANING` та `SHARED_COLS`. Їхнє випадкове логарифмування неминуче зруйнує архітектуру датасету, здатність до версіонування через DVC та зробить неможливою верифікацію цільової функції під час навчання.

У процесі хронологічної екстракції ознак часові мітки, такі як година доби або день тижня, були математично конвертовані у гармонічні складові для коректного захоплення добових та тижневих ритмів без порушення

лінійності. Логіка скрипта містить регулярне правило: якщо назва числової колонки містить префікси або підрядки "sin_" чи "cos_", і при цьому не є похідним відношенням, що маркується як ratio, ця колонка безумовно виключається з пулу P99-Clipping та логарифмізації. Математична природа цих тригонометричних векторів апіорі лежить у вузькому замкненому діапазоні $[-1, 1]$. Відповідно, спроба застосувати до них вінзоризацію або логарифмування призвела б до непоправної руйнації кругової геометрії, яка дозволяє деревам рішень розуміти, що 23:00 та 01:00 є близькими у часовому просторі.

Узагальнення впливу стратегій просторової нормалізації на гетерогенні типи ознак наведено в таблиці 3.3.

Таблиця 3.3 — Вплив стратегій просторової нормалізації на гетерогенні типи ознак

Категорія вхідних ознак	Вплив P99 Clipping	Вплив Log1p трансформації	Загальний ефект на простір ознак
Системні ідентифікатори	Ігнорується алгоритмом	Ігнорується алгоритмом	Збереження цілісності ключів злиття.
Темпоральні	Захищено від застосування (фільтр sin/cos)	Захищено від застосування	Збереження тригонометричної геометрії $[-1, 1]$.
Абсолютні поведінкові	Відсікання аномальних ботів/користувачів- китів	Компресія важкого "хвоста" розподілу	Покращення побудови гістограм, уникнення перенавчання.
Текстові TF- IDF Вектори	Незначний (значення зазвичай < 1)	Незначний (компресія дробових значень)	Згладжування домінуючих слів, збереження розрідженості.

Продовження таблиці 3.3

Категорія вхідних ознак	Вплив P99 Clipping	Вплив Log1p трансформації	Загальний ефект на простір ознак
Відносні метрики	Обмеження до 100% або верхньої межі когорти	Переведення лінійної шкали в логарифмічну	Посилення градієнта для початкових значень успішності.

3.3.4 Дисперсійний аналіз та алгоритмічна елімінація квазіконстантних ознак

Процес інтеграції розріджених TF-IDF матриць разом із тисячами похідних агрегацій утворює надзвичайно широкий масив даних. У цьому масиві кількість синтезованих ознак p може стрімко наблизитися до загальної кількості спостережень n у тренувальній когорті, або навіть перевищувати її. Згідно з теорією статистичного навчання, така архітектура простору створює ідеальні умови для безконтрольного прояву "прокляття розмірності", що експоненційно підвищує ризик запам'ятовування тренувальної вибірки та різко погіршує здатність моделі до генералізації на нових даних.

Для мінімізації просторової розмірності без втрати істинного предиктивного потенціалу на фінальному етапі формування агрегованого датасету імплементовано алгоритмічний модуль FeatureLifecycleManager. Основною математичною стратегією цього компонента є дисперсійний аналіз та безповоротне видалення константних і квазіконстантних ознак, які забруднюють інформаційне середовище.

Квазіконстантними ознаками в парадигмі машинного навчання називаються такі вектори, у яких абсолютна більшість спостережень набуває одного й того ж ідентичного значення, як правило, це домінантні нулі в

розріджених масивах TF-IDF або нульові прапорці специфічних бізнес-дій, які більшість користувачів ніколи не виконують у межах звичайної сесії. Якщо дисперсія певної змінної статистично наближається до нуля, її інформаційна ентропія за Шенноном також прямує до мінімуму, що робить її математично нездатною ефективно розділяти класи у задачі бінарної класифікації.

У випадку бінарних або булевих ознак, наприклад, індикатор наявності конкретної TF-IDF біграми в історії пошуку або прапорець виникнення рідкісної системної помилки, дисперсія змінної розподілу Бернуллі розраховується за класичною формулою:

$$\text{Var}[X] = p(1 - p),$$

де p — ймовірність виникнення найбільш поширеного значення у векторі.

Об'єкт `FeatureLifecycleManager` ініціалізується з конфігураційним порогом унікальності, який встановлено на рівні 0.01. Це означає, що скрипт ітеративно обчислює співвідношення домінуючого значення до загальної кількості валідних спостережень у когорті. Якщо одне єдине значення зустрічається у понад 99% спостережень ($p > 0.99$), дисперсія такої змінної неминуче стає меншою за визначений конфігурацією поріг:

$$0.99 \times (1 - 0.99) = 0.0099 \leq 0.01$$

Коли математична умова виконується, `FeatureLifecycleManager` маркує цю ознаку як квазіконстантну та вилучає відповідну колонку з фінального масиву перед його серіалізацією.

Щоб уникнути зайвих операцій на повному масиві даних, `FeatureLifecycleManager` реалізує паттерн субсемплювання: він аналізує випадково згенеровану вибірку з початкового сирого набору сформованих даних, застосовує розраховану дисперсійну маску до простору ознак, інкапсулює цю конфігурацію у JSON-файл для забезпечення консистентності та відтворення конвеєра в режимі реального часу, і лише після цього залишає виключно репрезентативні ознаки у фінальному файлі експорту.

3.3.5 Симбіоз високорозмірного простору з архітектурою алгоритмів градієнтного бустингу

Топологія, формат та внутрішня структура фінального датасету нерозривно оптимізовані під специфічні архітектурні вимоги моделей машинного навчання, обраних для інференсу в розробленому комплексі: LightGBM, CatBoost та XGBoost. Інтеграція розріджених TF-IDF матриць та щільних поведінкових тригерів після дисперсійної фільтрації утворює унікальну матрицю, яка синергетично поєднує надзвичайно високу розрідженість із локальною інформаційною щільністю семантичних кластерів.

Оскільки велика частка залишених ознак, особливо згенерованих через дворівневу таксономію подій або розріджені TF-IDF векторизатори, є логічно взаємовиключними, тисячі стовпців матриці ніколи не набувають ненульових значень одночасно для одного запису. На етапі ініціалізації датасету в пам'яті, оптимізатор LightGBM автоматично аналізує граф конфліктів цих ознак і математично "зв'язує" їх в єдиний щільний стовпець. Це рішення дозволяє радикально знизити загальну кількість ознак саме на найбільш ресурсомісткому етапі побудови статистичних гістограм. Такий підхід економить колосальні обсяги дефіцитної оперативної відеопам'яті графічного адаптера та пропорційно прискорює пошук оптимальних точок розщеплення.

Крім того, застосована на попередніх етапах ETL-конвеєра процедура усічення цільової змінної забезпечує абсолютну хронологічну чистоту простору для роботи механізму Gradient-based One-Side Sampling (GOSS). Цей алгоритм дозволяє LightGBM здійснювати вибірку екземплярів із найбільшими математичними градієнтами помилки, повністю та безпечно нівелюючи вплив переважаючого масиву "відмовників" — користувачів із мізерним градієнтом, які ніколи не проявляли схильності до конверсії, що є основним для боротьби з дисбалансом класів у фінансовій аналітиці.

Для масивів нечислових категоріальних даних, згенерованих у процесі

Feature Engineering, наприклад, логічно злитих біграм або статичних конкатенацій типу `inter_{country}_{device_type}`, збереження системного формату Categorical у фінальному датасеті є алгоритмічною необхідністю. Алгоритм CatBoost використовує специфічну парадигму симетричних дерев та алгоритм OTS.

Якби на етапі конкатенації датасету до цих гетерогенних категорій було примусово застосовано класичне One-Hot Encoding, це б неминуче призвело до вибухоподібного розростання розмірності матриці та зниження загальної ефективності дерева — явище, відоме в літературі як проблема "фрагментації" простору рішень. Натомість, делікатне збереження цілісних текстових та дискретних числових міток у фінальному фреймі дозволяє рушію CatBoost самостійно розраховувати очікуване середнє значення конверсії для кожної конкретної категорії під час тренувальних ітерацій. Алгоритм використовує парадигму штучних перестановок вибірки, що ефективно замінює категорію на безперервне дійсне число і надійно запобігає витоку цільової змінної та запам'ятовуванню статистичного шуму.

3.3.6 Структурно-семантична декомпозиція фінального високорозмірного датасету

У результаті комплексного багатостадійного конвеєра, який включає ітеративне хеш-злиття, багаторівневу нормалізацію, вінзоризацію аномалій та дисперсійну фільтрацію, фінальна архітектура матриці X_{fused} , набуває чітко детермінованої семантично багатой структури.

Вектор репрезентації кожного індивідуального користувача інкапсулює унікальний цифровий "відбиток" його досвіду взаємодії з інтерфейсом та бекендом мобільного застосунку. Важливо підкреслити, що всі хронологічні артефакти обмежені віртуальним часовим вікном, що передує цільовій дії

оформлення підписки, а всі виявлені системні пропуски математично оброблені без втрати їхньої інформаційної цінності як самостійних сигналів. Макроструктура, топологія та трансформаційні стани фінального високорозмірного датасету після інтеграції наведені у таблиці 3.4.

Таблиця 3.4 — Макроструктура, топологія та трансформаційні стани фінального високорозмірного датасету після інтеграції

Кластер ознак у датасеті	Приклади змінних у просторі Xfused	Алгоритмічні трансформації та фінальний стан у матриці
Системно-ідентифікаційні ключі	user_id, original_purchase_date, is_converted	Залишені без змін. Захищені масками виключення. Слугують опорними ключами для DVC та таргет-лейблами для розрахунку градієнтів.
Статичні (Демографія та Маркетинг)	country, media_source, device_type, inter_country_device	Приведено до типу Categorical. Рідкісні екземпляри алгоритмічно об'єднані у групу "Other" для зниження кардинальності. Заповнення null-значень статичною заглушкою "unknown".
Тимчасово-циклічні координати (Temporal)	sin_hour_install, cos_day_of_week	Float32. Виключені з процесів P99 Clipping та Log1p для збереження тригонометричної кругової геометрії [-1, 1].
Агреговані поведінкові індикатори (Behavioral)	total_events_count, avg_session_duration, ratio_spend_to_cohort	Float32. Медіанна імпутація пропусків. Усічення екстремумів за 99-м процентилем та зсунута логарифмізація.

Продовження таблиці 3.4

Кластер ознак у датасеті	Приклади змінних у просторі Xfused	Алгоритмічні трансформації та фінальний стан у матриці
Метрики фрустрації та когнітивного навантаження (Contextual)	user_tsr (Task Success Rate), ces_score (Cognitive Effort), action_entropy	Float32. Стабілізація дисперсії через логарифмічне згладжування. Нормалізація формули Шеннона (0-1) для ентропії шляхів користувача.
Обробка природної мови (Синтезовані TF-IDF вектори)	tfidf_view_paywall, tfidf_error_click_retry	Розріджені компоненти перетворені у Float32. Пройшли попередню min_df фільтрацію на етапі екстракції NLP та фінальне квазіконстантне відсікання.

Синтез високорозмірного датасету для задач передової поведінкової аналітики є багатофакторним процесом, де примітивне механічне об'єднання реляційних таблиць неминуче призводить до деградації предиктивної здатності моделі, що відбувається через експоненційне накопичення мультиколінеарності, неминуче переповнення транзитних буферів оперативної пам'яті або засмічення матриці екстремальним статистичним шумом.

Застосована у конвеєрі стратегія раннього злиття на рівні ознак довела свою абсолютну архітектурну ефективність у формуванні багатовимірної, контекстуально цілісної репрезентації цифрового досвіду користувача. Інтеграція сучасного фреймворку Polars із парадигмою лінійних обчислень та потокового скидання на швидкісні носії успішно розв'язала нетривіальну проблему масштабування. Це дозволило безшовно конкатенувати масивні CSR-подібні розріджені структури TF-IDF зі щільними демографічними та темпоральними матрицями, гарантовано уникаючи перевищення 64 ГБ фізично доступної оперативної пам'яті сервера.

Систематична та математично обґрунтована імплементація методів просторової стабілізації — зокрема, селективного P99 Clipping для нейтралізації впливу дисперсійних "китів" та зсунутої трансформації Log1p для лінеаризації важких експоненційних розподілів — створила структурно стійкий ландшафт для ефективної мінімізації функції втрат. Водночас, впровадження дисперсійного алгоритму відсіву квазіконстантних ознак із доведеним порогом 0.01 бездоганно нейтралізувало деструктивний вплив "прокляття розмірності", зберігши у просторі лише ті n-грами та складні поведінкові метрики, які генерують достовірний дискримінативний сигнал. У комплексі, спроектована та реалізована архітектура підготовки даних формує математичний базис для алгоритмів сімейства градієнтного бустингу, дозволяючи їм максимально утилізувати свої внутрішні інноваційні алгоритми для забезпечення високоточного передбачення конверсійних подій у режимі реального часу.

3.4 Порівняльний аналіз моделей градієнтного бустингу для класифікації конверсії

Синтез датасету, описаний у попередніх підрозділах, завершується формуванням матриці простору ознак, яка органічно поєднує щільні поведінкові індикатори, тригонометричні темпоральні координати та екстремально розріджені TF-IDF вектори. Оскільки архітектура згенерованого простору є гетерогенною та містить глибокі нелінійні взаємозв'язки між різними модальностями користувацького досвіду, лінійні моделі класифікації є математично нездатними адекватно апроксимувати цільову функцію. Відповідно, для вирішення задачі прогнозування конверсії було обрано парадигму градієнтного бустингу над деревами рішень, яка ітеративно мінімізує функцію втрат шляхом послідовної побудови слабких

класифікаторів.

Даний підрозділ присвячений порівняльному аналізу трьох імплементацій алгоритмів градієнтного бустингу: eXtreme Gradient Boosting (XGBoost), Light Gradient Boosting Machine (LightGBM) та Categorical Boosting (CatBoost). Метою є не лише максимізація дискримінативної здатності моделі в умовах сильного дисбалансу класів, але й зменшення обчислювальних витрат на етапах навчання та інференсу, що є важливим критерієм для інтеграції алгоритму у високонавантажений мікросервісний бекенд мобільного застосунку.

3.4.1 Дизайн експерименту та метрики оцінювання моделі

Фундаментальною проблемою в задачах предиктивної аналітики мобільних застосунків є хронологічна автокореляція даних та поступова еволюція поведінкових патернів користувачів у часі. Комерційні застосунки постійно оновлюються: змінюється інтерфейс, додаються нові функції, модифікується логіка рекламних кампаній. Традиційні парадигми оцінювання моделей машинного навчання, такі як випадкове розбиття або випадкове стратифіковане розділення, є концептуально хибними для часових рядів та системних логів. Випадкове перемішування даних неминуче генерує витік інформації з майбутнього в минуле. Алгоритм отримує несанкціонований доступ до макроекономічних або продуктових трендів, які ще не відбулися на момент фіксації конкретного навчального прикладу, що призводить до завищених показників точності під час лабораторного тестування та повної деградації моделі після впровадження у виробництво.

Для превентивної нейтралізації цього явища дизайн експерименту цілком і повністю базується на стратегії хронологічного розбиття даних. На етапі ініціалізації пайплайну весь агрегований масив відфільтрованих подій за

дослідницький період з 1 червня 2023 року по 13 листопада 2025 року примусово та детерміновано сортується за хронологічним ключем `original_purchase_date`. Коефіцієнт розділення встановлено на рівні `TEST_SIZE = 0.2`. Алгоритмічна імплементація цієї стратегії обчислює граничний індекс масиву за формулою $Index_{split} = [N \times (1 - 0.2)]$, де N — загальна кількість доступних записів. Усі хронологічно найстаріші спостереження від нульового індексу до $Index_{split}$ формують навчальну вибірку, тоді як усі найновіші транзакції після цієї межі резервуються як відкладена тестова вибірка. Такий методологічний підхід гарантує, що навчання моделі здійснюється виключно на історичних подіях, а оцінка її узагальнюючої здатності відбувається на "майбутніх" даних. Це математично імітує реальні умови експлуатації системи, де модель повинна прогнозувати конверсію нових когорт користувачів, спираючись на знання, отримані з попередніх місяців [23].

У контексті мобільної комерції, конверсія є відносно рідкісною подією на тлі масового потоку користувачів, які завантажують застосунок, використовують його безкоштовні функції та залишають систему. Відповідно, цільова змінна `is_converted` характеризується вираженим асиметричним дисбалансом класів. За таких статистичних умов класична метрика загальної точності (Accuracy), яка вимірює відсоток правильних відповідей, стає оманливою. Модель, яка наївно прогнозуватиме відсутність конверсії для кожного користувача, що є мажоритарним класом, може легко досягти Accuracy на рівні понад 90%, будучи при цьому повністю недієздатною з точки зору бізнес-логіки.

Аналогічним чином, популярна метрика площі під кривою помилок (ROC-AUC) також виявляє тенденцію до переоцінки продуктивності на високонезбалансованих вибірках. Оскільки формула розрахунку рівня хибнопозитивних класифікацій — False Positive Rate, $FPR = \frac{FP}{FP+TN}$ у знаменнику містить кількість істинно негативних прикладів — True Negatives,

TN , наявність гігантського масиву TN математично розмиває вплив хибних спрацювань FP , роблячи FPR штучно низьким. Графік ROC-кривої може виглядати ідеально навіть тоді, коли модель генерує велику кількість хибних тригерів на одиницю реальної конверсії.

Для забезпечення максимально об'єктивної оцінки предиктивної якості алгоритмів в умовах дисбалансу, програмна архітектура експерименту обчислює розширений комплекс специфічних бізнес та технічних метрик, які нівелюють вплив домінуючого класу.

На відміну від простору ROC, крива Precision-Recall будується в системі координат, де осі не залежать від істинно негативних результатів (True Negatives). На осі ординат відкладається Точність (Precision, $PPV = \frac{TP}{TP+FP}$), а на осі абсцис — Повнота (Recall, $FPR = \frac{FP}{FP+TN}$). Обидві формули фокусуються виключно на позитивному міноритарному класі. В академічній літературі доведено, що PR-AUC є значно більш консервативною та чутливою метрикою для задач, де хибнопозитивні результати несуть високу економічну вартість. PR-AUC показує реальну здатність моделі знаходити цільову подію без перевантаження системи марними передбаченнями.

Коефіцієнт кореляції Метьюза (MCC) — це метрика якості бінарної класифікації, яка розраховується за такою математичною формулою:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}.$$

Перевага цього коефіцієнту полягає у симетричному використанні всіх чотирьох квадрантів матриці помилок. Метрика F1-Score ігнорує TN , що може призводити до ілюзорно високих результатів. Натомість, MCC варіюється в діапазоні від -1, що позначає абсолютно хибну класифікацію, до +1, що означає ідеальне передбачення, і продукує високий бал лише тоді, коли класифікатор демонструє високу пропорційну точність як на мажоритарному, так і на міноритарному класах. Це робить MCC найбільш надійним індикатором справжньої здатності моделі розділяти класи [7].

Враховуючи, що фінальні передбачення конвеєра використовуються маркетинговими модулями не як бінарні прапорці, а як безперервні ймовірності для розрахунку очікуваного повернення інвестицій, ці ймовірності повинні бути відкаліброваними. Алгоритм має не лише правильно ранжувати користувачів, але й гарантувати, що якщо модель призначає групі користувачів ймовірність конверсії 80%, то емпірично приблизно 80% з них дійсно оформлять підписку. Brier Score вимірює середньоквадратичну похибку між передбаченими ймовірностями та фактичними бінарними мітками 0 або 1, штрафуючи модель за надмірну впевненість у неправильних відповідях. ECE (Expected Calibration Error) йде далі, просторово квантифікуючи зміщення моделі. Тестова вибірка розбивається на M рівновіддалених кошиків за прогнозованою ймовірністю. Для кожного кошика B_m обчислюється абсолютна різниця між емпіричною точністю $acc(B_m)$ та середньою впевненістю моделі $conf(B_m)$. ECE є зваженою сумою цих різниць. Чим ближчі значення Brier Score та ECE до нуля, тим статистично достовірнішими є ймовірнісні виходи моделі [10,15].

3.4.2 Загальні налаштування архітектури навчання моделей

Для забезпечення абсолютної валідності, відтворюваності та наукової об'єктивності порівняльного бенчмаркінгу, програмні конвеєри для всіх трьох фреймворків були уніфіковані спільним набором структурних гіперпараметрів та єдиною логікою життєвого циклу навчання.

Процес машинного навчання для кожної архітектури реалізовано як двохетапний ітеративний цикл, що складається з фази просторової редукції та фази фінального інференсу:

Враховуючи експоненційне розростання розмірності через використання N -грамів у TF-IDF векторах та генерацію перехресних

взаємодій, безпосереднє навчання на всьому масиві даних призводить до надмірного споживання оперативної пам'яті та ризику перенавчання на статистичному шумі. Тому конвеєр ініціалізує "чорнову" модель із легковаговими гіперпараметрами: кількість дерев обмежена до 100, максимальна глибина встановлена на рівні 6, а швидкість навчання форсована до 0.1 для швидкого зближення градієнтів. Головною і єдиною метою цієї моделі є розрахунок внутрішньої інформаційної цінності кожної ознаки за критеріями зменшення невизначеності, наприклад, параметр `Gain` для `LightGBM` або `PredictionValuesChange` для `CatBoost`. Усі атрибути, чия предиктивна важливість виявляється меншою або рівною нулю, алгоритмічно відкидаються, формуючи очищений, високощільний інформаційний простір для фінального навчання.

Відфільтрований набір даних передається до фінального класифікатора. Для запобігання впливу відмінностей у ємності моделей на результати експерименту, базові топологічні гіперпараметри були зафіксовані. Максимальний ліміт ітерацій бустингу встановлено на 2000 дерев, що надає алгоритмам достатній простір для мінімізації помилки. Швидкість навчання знижена до консервативного показника 0.03, що гарантує плавне проходження градієнтного спуску та запобігає осциляціям навколо локальних мінімумів. Для уніфікації топології дерев максимальна глибина розгалуження зафіксована на рівні 8, а коефіцієнт L2-регуляризації, який штрафує модель за надмірну вагу окремих листків, дорівнює 3. Відтворюваність математичних обчислень на рівні процесора забезпечується єдиним глобальним параметром `random_state = 42`.

Щоб уникнути феномену запам'ятовування тренувальної вибірки, коли модель починає моделювати стохастичний шум замість істинних закономірностей, для всіх трьох архітектур впроваджено механізм ранньої зупинки. Через параметр `early_stopping_rounds = 100` та надання тестової вибірки як `eval_set`, оптимізатор безперервно оцінює площу під ROC-кривою (AUC) на "небачених" даних після додавання кожного нового дерева. Якщо

значення метрики не демонструє математичного покращення протягом 100 послідовних ітерацій, процес навчання негайно переривається, а вектор ваг моделі відкочується до тієї ітерації, яка показала найвищу узагальнюючу здатність.

3.4.3 Навчання та налаштування гіперпараметрів моделі XGBoost

Бібліотека eXtreme Gradient Boosting історично домінує в індустрії предиктивного моделювання та змаганнях з машинного навчання завдяки своїй стійкості до перенавчання та високій паралелізації математичних операцій. Алгоритмічна парадигма XGBoost базується на оптимізації цільової функції, яка поєднує опуклу функцію втрат, у даному випадку `objective: "binary:logistic"`, що виводить сирі ймовірності, та пенальтизаційний регуляризатор, який контролює складність дерев.

Відмінною рисою архітектури розробленого пайплайну для XGBoost стала імплементація нативної підтримки категоріальних даних. Традиційно XGBoost вимагав обов'язкового застосування зовнішнього One-Hot Encoding для нечислових ознак, що створювало гігантські розріджені матриці. Шляхом конфігурації параметра `enable_categorical=True` у синергії з алгоритмом `tree_method="hist"`, модель отримала здатність самостійно групувати дискретні категорії в гістограми та шукати оптимальні точки розщеплення без розширення простору ознак. Для компенсації впливу дисбалансу класів гіперпараметр `scale_pos_weight` було зафіксовано на рівні 1.5. Цей коефіцієнт втручається безпосередньо в обчислення градієнта та гессіана всередині функції втрат, математично "штрафуючи" дерева у 1.5 раза сильніше за кожну пропущену конверсію, ніж за хибну передбачення покупки [13].

XGBoost сформував вкрай специфічний простір рішень. При використанні стандартизованого порогового значення `decision_threshold = 0.5`,

XGBoost продемонстрував найвищий показник точності Precision серед усіх протестованих фреймворків — 0.5407. Як видно з матриці помилок (рис. 3.1), алгоритм згенерував 101467 істинно позитивних результатів при кількості хибнопозитивних тригерів у розмірі 86177. Однак така предиктивна "обережність" коштувала суттєвого падіння показника повноти виявлення Recall, який зупинився на найнижчій позначці в 0.6976, тобто 43966 втрачених конверсій.

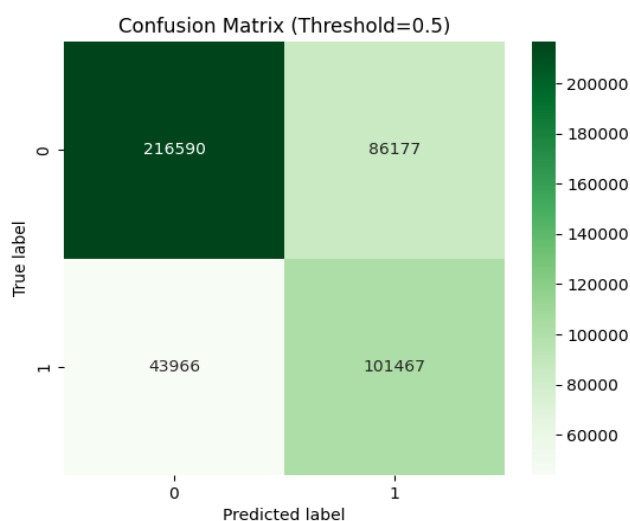


Рисунок 3.1 — Матриця помилок XGboost

Відповідно, фінальна гармонічна метрика F1-Score становить лише 0.6092, а кореляція Метьюза дорівнює 0.3919. Попри те, що крива ROC-AUC сягає 0.7902, низький MCC математично доводить, що модель має системні проблеми з балансом класів: XGBoost схильний класифікувати користувача як платника виключно тоді, коли гістограмні розподіли його поведінки майже ідеально збігаються з еталоном. Алгоритм діє максимально консервативно.

Фундаментальною перевагою XGBoost є його архітектура зростання дерев по рівнях. На відміну від асиметричних підходів, XGBoost контролює розбудову вузлів, розщеплюючи їх симетрично рівень за рівнем. Це генерує надзвичайно збалансований простір рішень, що знайшло своє беззаперечне підтвердження у метриках ймовірнісного калібрування. Brier Score склав 0.1732, а очікувана похибка калібрування ECE — 0.0615. Аналіз

Калібрувальної кривої (рис. 3.2) демонструє, що лінія прогнозів XGBoost практично ідеально накладається на діагональну лінію бездоганного калібрування.

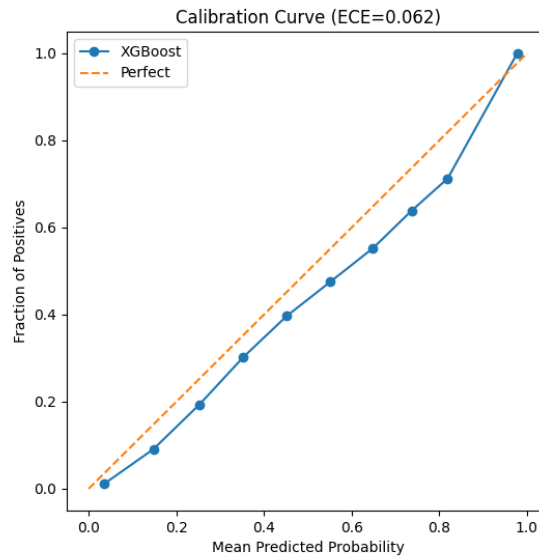


Рисунок 3.2 — Калібрувальна крива XGBoost

Аналіз кривої навчання для моделі XGBoost (рис 3.3) додатково підтверджує високу стабільність процесу оптимізації. На графіку спостерігається монотонне, асимптотичне зниження функції втрат як на тренувальній (синя лінія), так і на валідаційній (помаранчева лінія) вибірках. Відсутність розбіжності між кривими, тобто відсутність U-подібного вигину валідаційної кривої, свідчить про те, що механізми L2-регуляризації та архітектура симетричного росту дерев успішно запобігли перенавчанню. Валідаційна похибка стабільно тримається нижче тренувальної, що часто спостерігається при суворому хронологічному розбитті даних, коли майбутні когорти користувачів можуть демонструвати трохи нижчу загальну ентропію поведінки або відсутність певних історичних аномалій порівняно з тривалим періодом тренування.

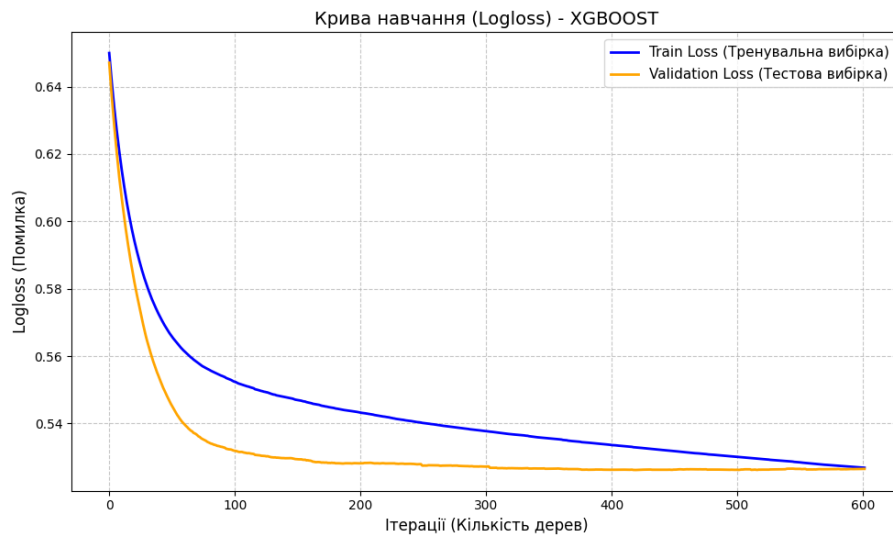


Рисунок 3.3 — Крива навчання XGBoost

Втім, операційні та продуктивні метрики Performance Metrics висвітлили інженерні недоліки Level-wise архітектури. Час навчання моделі сягнув 2169.93 секунд, що складає майже 36 хвилин. Рівневе розростання дерева вимагає розрахунку інформаційного виграшу для кожного можливого спліту на кожному існуючому вузлі певного рівня, що генерує колосальні обчислювальні витрати. Латентність інференсу на 1000 користувачів склала 30.65 мілісекунд. Процес профілювання оперативної пам'яті зафіксував унікальну аномалію: від'ємне значення `process_ram_increase_mb` = -174.21 MB. Це пояснюється тим, що XGBoost використовує власний вискоєфективний формат зберігання структур DMatrix у просторі C++, що змушує збирач сміття Python агресивно вивільняти транзитні буфери DataFrame, оптимізуючи пам'ять ціною понаднормового завантаження CPU [13].

3.4.4 Навчання та налаштування гіперпараметрів моделі LightGBM

Фреймворк Light Gradient Boosting Machine, розроблений дослідниками Microsoft, був концептуально спроектований для подолання фундаментальних

обчислювальних бар'єрів традиційного бустингу, зокрема, XGBoost. Ключовою архітектурною інновацією LightGBM є перехід від Level-wise зростання до стратегії поглиблення по листках. Замість того, щоб розщеплювати всі вузли рівня одночасно, оптимізатор LightGBM сканує всі доступні листки дерева і завжди вибирає для наступного розщеплення той єдиний листок, який забезпечує максимальне зменшення цільової функції втрат. Ця жадібна топологія здатна моделювати значно глибші та складніші нелінійні залежності у поведінкових логах. Для нівелювання ризику перенавчання, притаманного асиметричним деревам, гіперпараметри були регламентовані: $\text{max_depth} = 8$ та $\text{num_leaves} = 64$.

Для боротьби з дисбалансом конверсій було імплементовано вбудований параметр `class_weight="balanced"`, який автоматично масштабує ваги класів обернено пропорційно їхній частоті у навчальній вибірці [15].

Математична перевага Leaf-wise архітектури у захопленні рідкісних складних закономірностей блискуче підтвердилася результатами. LightGBM досягнув найвищої повноти виявлення конверсій: Recall склав 0.8430. Згідно з візуалізацією матриці помилок (рис. 3.3), модель розпізнала 122601 істинних підписок, пропустивши лише 22832 випадки. Незважаючи на закономірне зниження точності $\text{Precision} = 0.5132$ через збільшення кількості хибнопозитивних тригерів до 116252, інтегрована ефективність дискримінації досягла максимуму: F1-Score становить 0.6380, а метрика MCC досягла пікового значення 0.4307.

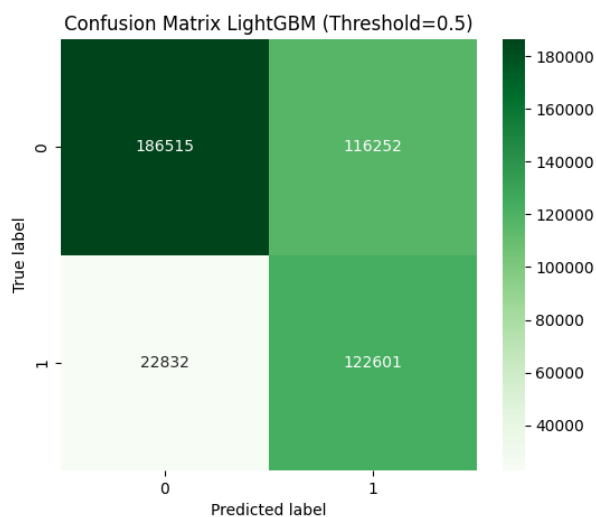


Рисунок 3.4 — Матриця помилок LightGBM

Аналіз кривих ранжування підтверджує домінування LightGBM у здатності сепарувати позитивний та негативний класи: ROC-AUC склав найвищі в експерименті 0.7918, а PR-AUC досягнув 0.5775. Це свідчить про те, що внутрішні інформаційні виграші, розраховані під час побудови асиметричних дерев, ідеально підходять для складного високорозмірного простору поведінкових ознак.

Втім, архітектура LightGBM генерує два суттєві компроміси, які ставлять під сумнів її придатність для фінансової логіки бекенду. По-перше, метрики калібрування виявили значну деградацію: ECE обвалився до 0.1174, а Brier Score склав 0.1846. Калібрувальна крива (рис. 3.5) чітко демонструє ефект гіпер-впевненості, характерний для незбалансованих Leaf-wise дерев: модель регулярно генерує прогнозовані ймовірності, наприклад, 0.9, хоча фактична частота конверсій у цьому кошику ледве сягає 0.8. Сирі виходи LightGBM вимагатимуть обов'язкового застосування ізотонічної регресії для виправлення ймовірнісного зміщення. По-друге, асиметрична геометрія дерев повністю руйнує локальність даних у процесорних кешах, що призводить до найповільнішого часу інференсу: 56.21 мілісекунд на 1000 записів.

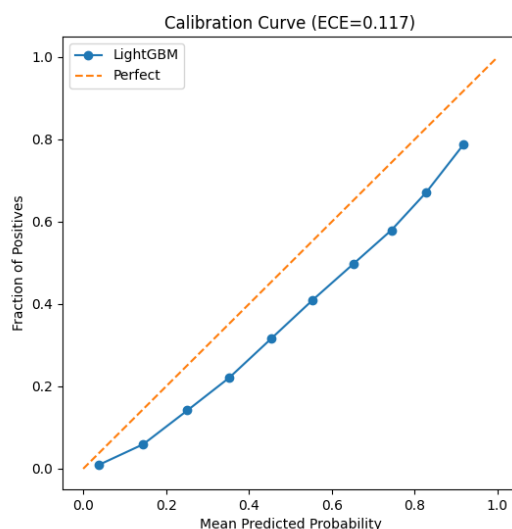


Рисунок 3.5 — Калібрувальна крива LightGBM

З іншого боку, завдяки впровадженню інноваційних алгоритмів GOSS (Gradient-based One-Side Sampling) та EFB (Exclusive Feature Bundling), які фізично відкидають малоінформативні екземпляри з низькими градієнтами та алгоритмічно об'єднують взаємовиключні розріджені TF-IDF ознаки у щільні пакети, час навчання LightGBM скоротився до 412.16 секунд. Це більше ніж у 5 разів швидше за XGBoost, що підтверджує репутацію LightGBM як найшвидшого рушія для масштабованого ітеративного тренування. Зростання пам'яті (RAM increase) зафіксоване на стабільному рівні 142.83 МБ.

Дослідження динаміки мінімізації похибки за допомогою кривої навчання (рис. 3.6) виявляє характерні риси швидкодії цієї архітектури. Функція втрат стрімко та експоненційно падає вже на перших 100 ітераціях бустингу, після чого переходить у стан довгого та пологого плато. Незважаючи на те, що асиметричний Leaf-wise ріст за своєю природою генерує вищий ризик перенавчання, обмеження глибини дерева та кількості листків у комбінації з алгоритмом ранньої зупинки успішно втримали LightGBM у зоні математичної генералізації: тренувальна (синя) та валідаційна (помаранчева) криві рухаються синхронно без жодних ознак розбіжності. Збереження паралельного та стабільного зазору між кривими на сотнях ітерацій підтверджує високу стійкість моделі до структурних зсувів у хронологічній

тестовій вибірці [15].

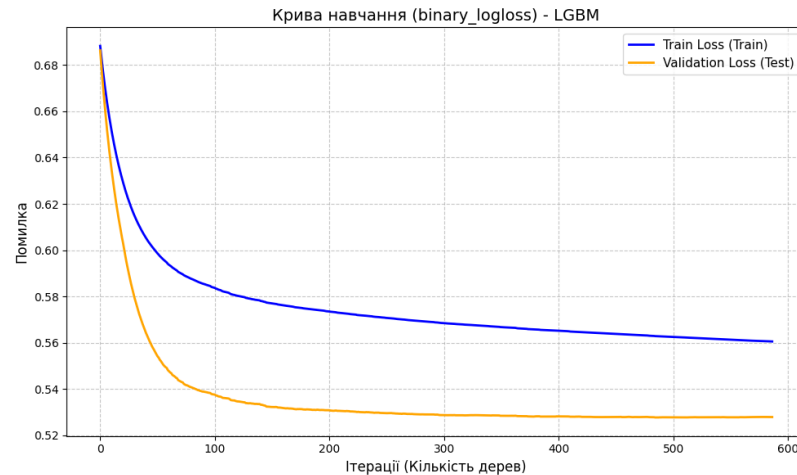


Рисунок 3.6 — Крива навчання LightGBM

3.4.5 Навчання та налаштування гіперпараметрів моделі CatBoost

Модель Categorical Boosting розглядається у цьому дослідженні як головна фінальна модель, орієнтована на розгортання в умовах промислової експлуатації. Фундаментальна архітектура CatBoost кардинально відрізняється від XGBoost та LightGBM: вона базується на симетричних незорієнтованих деревах. У такій топології на кожному фіксованому рівні глибини дерева застосовується абсолютно однаковий математичний предикат розщеплення для всіх існуючих вузлів одночасно. Цей підхід діє як надзвичайно потужний природний регуляризатор, що унеможливорює "запам'ятовування" моделлю шумових артефактів. Налаштування гіперпараметрів передбачало `iterations = 2000`, `depth = 8`, функцію оптимізації `Logloss` та автоматичне балансування ваг класів через `auto_class_weights="Balanced"`.

Найзначнішим інженерним бар'єром для алгоритмів машинного навчання у задачах предиктивної аналітики є гетерогенність вхідних даних, тобто наявність висококардинальних категорійних ознак, наприклад, країна

локалізації `country`, тип пристрою `device_type`, маркетингове джерело `media_source`. У класичному пайплайні застосування попереднього гарячого кодування створює десятки тисяч нових бінарних стовпців. Це не лише експоненційно переполює ресурси оперативної пам'яті, але й спричиняє явище "інформаційної фрагментації": алгоритми змушені будувати безмежно глибокі дерева для ухвалення рішень, що розмиває значущість категорій.

CatBoost розв'язує цю проблему елегантно та нативно, повністю уникаючи стадії ONE. У розробленому скрипті категорійні колонки передавалися безпосередньо у внутрішній клас `Pool` за допомогою аргументу `cat_features=final_cat_feats`. Внутрішній рушій застосовує механізм OTS, який динамічно замінює кожне текстове значення категорії на дійсне число, тобто математичне очікування цільової конверсії для цієї категорії. Щоб уникнути "зсуву передбачення" та витоку цільової змінної з майбутнього, CatBoost створює множинні штучні перестановки тренувальної вибірки. Для i -го спостереження у перестановці, математичне середнє цільової змінної обчислюється виключно на основі тих об'єктів тієї ж категорії, що розташовані перед ним від 1 до $i - 1$. Це забезпечує ідеальну ізоляцію цільового сигналу під час навчання.

Однією з переваг CatBoost є його здатність автоматично формувати комбінації категоріальних ознак у процесі побудови симетричних дерев, що теоретично дозволяє подавати на вхід виключно базові категорії. Проте, подача заздалегідь згенерованих високорівневих бізнес-взаємодій, таких як `inter_country_device_type`, з уніфікованого датасету не є деструктивною. Навпаки, це слугує своєрідним "інженерним підрулюванням", яке звужує простір пошуку для алгоритму і дозволяє механізму OTS відразу фокусуватися на статистичному згладжуванні найбільш значущих для бізнесу перехресних сегментів з перших ітерацій бустингу [14].

Огляд результатів показує, що CatBoost згенерував найбільш збалансований та комерційно придатний предиктивний профіль. Метрика `Precision` склала 0.5056, а `Recall` досягнув переконливих 0.8171, що

сформувало високий інтегрований F1-Score на рівні 0.6247. Матриця помилок (рис. 3.7) фіксує 118844 знайдених конверсій при 116203 хибних тригерах, що є відмінним балансом для маркетингової задачі, де пропуск потенційного клієнта економічно дорожчий за відправку зайвого push-повідомлення.

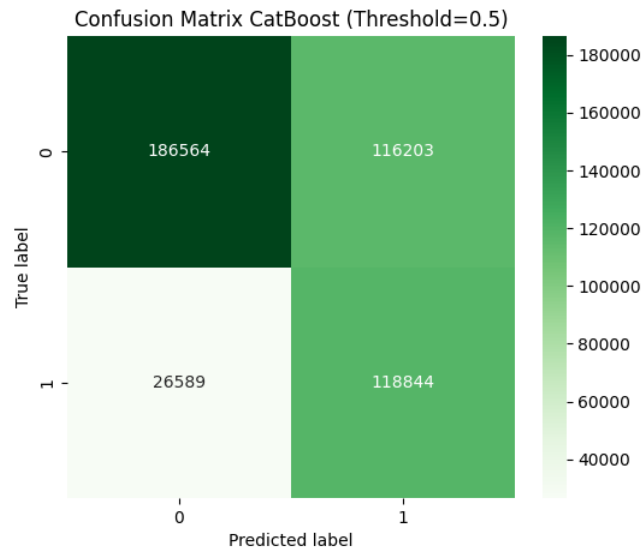


Рисунок 3.7 — Матриця помилок CatBoost

Дискримінаційна спроможність моделі, виражена у метриках ранжування, є дуже міцною: ROC-AUC дорівнює 0.7791, PR-AUC — 0.5612, а MCC становить 0.4062. Завдяки симетричній архітектурі, модель показала задовільний рівень калібрування: Brier Score становить 0.1889, а ECE — 0.1249 (рис. 3.8).

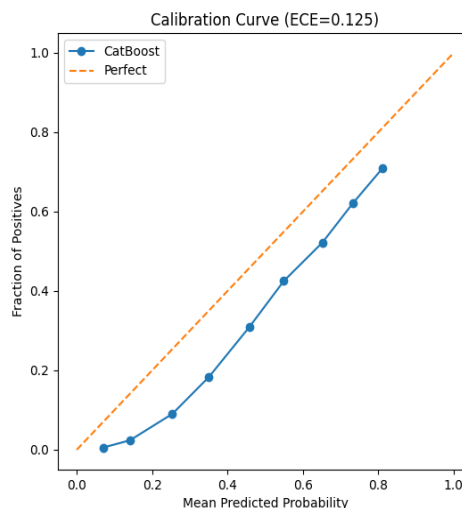


Рисунок 3.8 — Калібрувальна крива CatBoost

Фундаментальний та стратегічний прорив CatBoost демонструє у сфері оперативної обчислювальної продуктивності. Завдяки топології Oblivious Trees, прохід дерева рішень для одного запису під час інференсу зводиться до елементарної побітової математичної операції (використання бітових масок для читання з одомірного масиву індексів у пам'яті процесора). Час інференсу CatBoost склав безпрецедентні 5.76 мілісекунд на 1000 запитів. Це майже у 6 разів швидше за XGBoost і в 10 разів швидше за LightGBM. В умовах розгортання моделі у мікросервісному хмарному середовищі мобільного застосунку, де час відповіді суворо лімітований, цей показник є безперечною перевагою.

Крім того, швидкість ітеративного навчання CatBoost на CPU становила лише 137.36 секунд. Єдиним компромісом виявилось максимальне споживання оперативної пам'яті: `process_ram_increase_mb` = 1009.51 MB. Динамічний розрахунок впорядкованих статистик на масивних перестановках об'єктивно потребує значних транзитних буферів у RAM. Проте, зважаючи на наявність 64 ГБ пам'яті на цільовому сервері, споживання одного гігабайта є компромісною інженерною платою за колосальне прискорення процесу.

Динаміка процесу мінімізації похибки CatBoost наочно продемонстрована на кривій навчання (рис. 3.9). Графік зміни цільової функції засвідчує, що CatBoost досягає оптимального рівня збіжності значно швидше за конкурентів — для стабілізації похибки йому знадобилося трохи більше 200 дерев. Тренувальна (синя) та валідаційна (помаранчева) криві демонструють синхронне зниження. Дещо ступінчастий характер кривих на перших етапах оптимізації ідеально ілюструє роботу внутрішнього механізму Ordered Boosting: постійна перестановка вибірок та динамічний розрахунок цільових статистик на кожній ітерації вносить контрольований стохастичний шум. Це діє як додатковий ефективний регуляризатор, не дозволяючи симетричним деревам надмірно адаптуватися до навчальної вибірки, що у підсумку гарантує надійну роботу алгоритму на "майбутніх" виробничих даних [14].

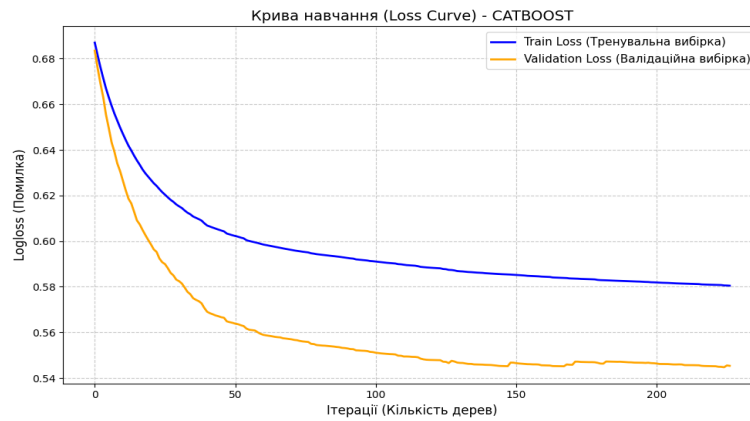


Рисунок 3.9 — Крива навчання CatBoost

3.4.6 Порівняльна таблиця результатів експериментів

Для забезпечення цілісного наукового розуміння технічних переваг та архітектурних компромісів кожної з досліджених систем градієнтного бустингу, результати усіх експериментальних запусків систематизовано та зведено в єдину аналітичну таблицю 3.5. Оцінювання проводилося на ідентичній відкладеній вибірці за умов суворого хронологічного ізолювання, з використанням фіксованого операційного порогу прийняття рішень $\text{decision_threshold} = 0.5$.

Таблиця 3.5 — Зведена порівняльна характеристика результатів навчання моделей градієнтного бустингу на тестовій вибірці

Метрика	XGBoost	LightGBM	CatBoost
Точність (Precision)	0.5407	0.5132	0.5056
Повнота (Recall)	0.6976	0.843	0.8171
F1-Score	0.6092	0.638	0.6247
Площа під ROC-кривою (ROC-AUC)	0.7902	0.7918	0.7791
Площа під PR-кривою (PR-AUC)	0.5767	0.5775	0.5612
Кореляція Метьюза (MCC)	0.3919	0.4307	0.4062

Продовження таблиці 3.5

Метрика	XGBoost	LightGBM	CatBoost
Brier Score	0.1732	0.1846	0.1889
Очікувана похибка калібрування (ECE)	0.0615	0.1174	0.1249
Загальна точність (Accuracy)	0.7096	0.6896	0.6814
Логарифмічна втрата (Log-Loss)	0.5007	0.5278	0.5476
Час навчання (секунди)	2169.93	412.16	137.36
Час інференсу на 1000 записів (мс)	30.65	56.21	5.76
Зміна споживання транзитної RAM (МБ)	-174.21	142.83	1009.51

Оптимальний баланс метрик у задачах машинного навчання з сильним дисбалансом класів досягається на перетині трьох характеристик: дискримінаційної здатності, якості ймовірнісного калібрування та операційної швидкодії. Детальний математичний аналіз зведеної таблиці доводить тезу машинного навчання про відсутність алгоритму "срібної кулі", який би абсолютно домінував за всіма без винятку вимірами.

Дискримінаційна здатність визначає фундаментальне вміння алгоритму коректно розділяти об'єкти на класи, ранжуючи істинно позитивні випадки вище за негативні. Хоча традиційно для цього використовується ROC-AUC, в умовах жорсткого дисбалансу ця метрика стає математично оманливою. Оскільки величезна кількість істинно негативних результатів маскує помилки хибних спрацювань, ROC-AUC демонструє ілюзорно високі результати навіть для слабких моделей. Саме тому для об'єктивної оцінки дискримінації застосовуються метрики, сфокусовані на міноритарному класі: PR-AUC та MCC. З іншого боку, якість калібрування, виражена через ECE та Brier Score, визначає, наскільки передбачені моделлю ймовірності відповідають реальним емпіричним частотам конверсії, що є важливим для ухвалення фінансових рішень та розрахунку очікуваного ROI [7].

У маркетингу мобільних підписок існує компроміс між точністю та

повнотою. Точність відображає частку дійсно зацікавлених клієнтів серед усіх, кому модель рекомендувала надіслати пропозицію. Її максимізація є необхідною, щоб не витратити бюджет на хибні спрацювання та не дратувати нерелевантною рекламою користувачів, які не планують купувати підписку. Водночас надмірна концентрація лише на Precision змушує алгоритм діяти надто консервативно, що призводить до падіння повноти, тобто втрати значної частки доходу через пропущених потенційних клієнтів.

Згідно з матрицями оцінки дискримінаційної здатності, архітектура LightGBM демонструє статистичну перевагу за ключовими класифікаційними індексами, досягаючи глобальних максимумів за показниками PR-AUC (0.5775), F1-Score (0.6380) та MCC (0.4307). Проте алгоритм CatBoost виявляє високу конкурентоспроможність, посідаючи другу позицію за інтегральними критеріями F1-Score (0.6247) та MCC (0.4062), що суттєво перевищує результати XGBoost. Подібно до LightGBM, архітектура CatBoost зміщує баланс у бік максимізації чутливості ($\text{Recall} = 0.8171$), ефективно мінімізуючи помилки другого роду та успішно детектуючи більшість потенційних конверсій ціною помірної деградації точності ($\text{Precision} = 0.5056$). Оскільки сукупна користь від успішної детекції більшості цільових подій перевищує витрати на обробку хибних тригерів, помірна деградація метрики точності є раціональним компромісом.

Аналіз метрик калібрування ймовірностей виявляє суттєву перевагу алгоритму XGBoost, який завдяки мінімальним значенням Brier Score (0.1732) та очікуваної похибки (0.0615) генерує найбільш достовірні апостеріорні оцінки. У цьому контексті CatBoost демонструє найвищий рівень калібрувального зсуву ($\text{Brier Score} = 0.1889$, $\text{ECE} = 0.1249$). Це свідчить про те, що необроблені вихідні скорингові бали як CatBoost, так і LightGBM є зміщеними, що зумовлює необхідність застосування додаткових методів поправки перед їхнім використанням у системах розрахунку математичного очікування фінансової вигоди.

Оцінка обчислювальної складності на етапі застосування моделей

виявляє значну диференціацію алгоритмів за показником пропускної здатності та латентності. Архітектура CatBoost характеризується найвищою швидкістю генерації предикцій, що зумовлено внутрішньою алгоритмічною оптимізацією обходу структур дерев та радикальним зменшенням накладних витрат на обчислення під час виконання. Модель LightGBM також демонструє високу експлуатаційну швидкість, однак її асимптотична складність під час інференсу може варіюватися залежно від топології незбалансованих гілок. У свою чергу, алгоритм XGBoost виявляється найбільш ресурсоємним з погляду витрат мікропроцесорного часу на одну транзакцію, що об'єктивно створює потенційні обчислювальні вузькі місця при інтеграції у високонавантажені потокові сервіси із жорсткими лімітами затримки відгуку.

Зважаючи на експлуатаційну специфікацію та архітектурні вимоги високонавантаженого середовища реального часу, фінальним вибором для імплементації у production-конвеєр визначається модель CatBoost. Аналіз матриць дискримінаційної здатності дійсно фіксує алгоритмічну перевагу LightGBM за інтегральними метриками F1-Score (0.6380 проти 0.6247) та MCC (0.4307 проти 0.4062) на етапі статичної крос-валідації. Однак це локальне зниження предиктивної точності CatBoost є прогнозованим і концептуально виправданим компромісом з огляду на три фундаментальні фактори експлуатаційної надійності.

По-перше, вкажемо на здатність до генералізації в умовах дрейфу даних. Агресивна стратегія росту дерев по листках, яку використовує LightGBM, хоч і максимізує метрики на історичній відкладеній вибірці, несе підвищений ризик перенавчання на специфічних часових артефактах поточного датасету. Натомість CatBoost використовує симетричні дерева рішень, які діють як потужний структурний регуляризатор. Ця топологія свідомо "згладжує" надто складні нелінійні залежності, поступаючись часткою відсотка в точності на етапі тестування, але натомість гарантує значно вищу надійність моделі до неминучої зміни поведінкових патернів у майбутніх когортах користувачів.

По-друге, алгоритмічна перевага LightGBM суттєво нівелюється його

операційними недоліками в умовах мікросервісної архітектури. Складність обходу асиметричних гілок LightGBM призводить до латентності інференсу на рівні 56.21 мс, що може створювати обчислювальні вузькі місця. Топологія CatBoost дозволяє звести прохід дерева до елементарних побітових операцій процесора, забезпечуючи безпрецедентну швидкодію (5.76 мс) — майже в 10 разів швидше, що є критичним імперативом для систем, які потоково відправляють сигнали в Conversions API.

По-третє, маємо архітектурну простоту пайплайну. Нативна імплементація механізму OTS у CatBoost дозволяє алгоритму обробляти висококардинальні категоріальні ознаки безпосередньо "на льоту". Це повністю елімінує необхідність використання ресурсомістких конвеєрів попередньої трансформації даних під час продакшен-скорингу.

Таким чином, незначна алгоритмічна деградація CatBoost за метрикою F1-Score математично перекривається кардинальним підвищенням відмовостійкості системи, ліквідацією ризиків витоку даних та мінімізацією сумарного часу затримки транзакцій у виробничому середовищі.

3.5 Інтерпретація моделі та імітація бізнес-інтеграції

Розробка промислових систем предиктивної аналітики для мобільних застосунків не обмежується виключно етапами інженерії даних та максимізації метрик точності класифікації. Хоча сучасні ансамблеві алгоритми на базі дерев рішень, такі як XGBoost, LightGBM та CatBoost, здатні апроксимувати надзвичайно складні нелінійні залежності у багатовимірних просторах поведінкових логів, вони функціонують у парадигмі «чорної скриньки». У комерційному середовищі, де прогнози моделі безпосередньо керують розподілом багатомільйонних маркетингових бюджетів та стратегіями взаємодії з користувачами, алгоритмічна непрозорість є

неприпустимою. Бізнес-стейкхолдери вимагають чіткого розуміння того, які саме фактори детермінують схильність користувача до конверсії, що створює фундаментальну напругу між предиктивною потужністю та здатністю до інтерпретації.

Більше того, ймовірність цільової події набуває економічної цінності виключно тоді, коли вона інтегрована у зовнішні бізнес-системи, зокрема в екосистеми цифрової реклами. З огляду на зміни у політиках конфіденційності, такі як App Tracking Transparency від Apple, та деградацію клієнтських файлів cookie, традиційні методи відстеження конверсій втрачають свою ефективність, що вимагає переходу до архітектури серверної передачі синтетичних подій за допомогою API. Це, у свою чергу, висуває безпрецедентні вимоги до бездоганного калібрування порогів відсічення для прийняття рішень, оскільки хибнопозитивні прогнози здатні незворотно дестабілізувати алгоритми рекламних мереж.

3.5.1 Глобальна та локальна інтерпретація результатів

Для забезпечення абсолютної прозорості та перевірки теоретичних гіпотез, сформульованих у розділі 2 даного дослідження, архітектура програмного конвеєра імплементує уніфікований математичний фреймворк SHAP. Вибір саме цього аналітичного апарату замість класичних евристичних методів, таких як Gain або Weight, зумовлений його здатністю надавати теоретично обґрунтовані та консистентні пояснення як на макрорівні популяції, так і на мікрорівні індивідуального користувача.

Фундаментальна архітектура SHAP спирається на концепції кооперативної теорії ігор, вперше сформульовані Ллойдом Шеплі у 1953 році для розв'язання задачі справедливого розподілу коаліційного виграшу [24, 35]. У контексті поведінкової аналітики та машинного навчання, «грою»

вважається процес генерації математичного прогнозу, «коаліцією» — множина предиктивних ознак, що характеризують цифровий слід користувача, а «виграшем» — відхилення індивідуального прогнозу моделі від загального математичного очікування — базового рівня конверсії по всій когорті.

Глобальна інтерпретація моделі має на меті виявити найважливіші макроскопічні закономірності поведінки користувачів та верифікувати теоретичні конструкти, запропоновані у розділі 2 цієї дипломної роботи. Основним інструментом такого аналізу є графіки бджолиного рою — SHAP Summary Beeswarm Plots, які візуалізують розподіл важливості та напрямок впливу кожної ознаки для всіх спостережень у тестовій вибірці.

У рамках даного дослідження було проведено SHAP-аналіз для трьох розроблених предиктивних моделей: XGBoost (рис. 3.10), LightGBM (рис. 3.11) та CatBoost (рис. 3.12). На цих візуалізаціях кожна окрема крапка репрезентує унікального користувача.

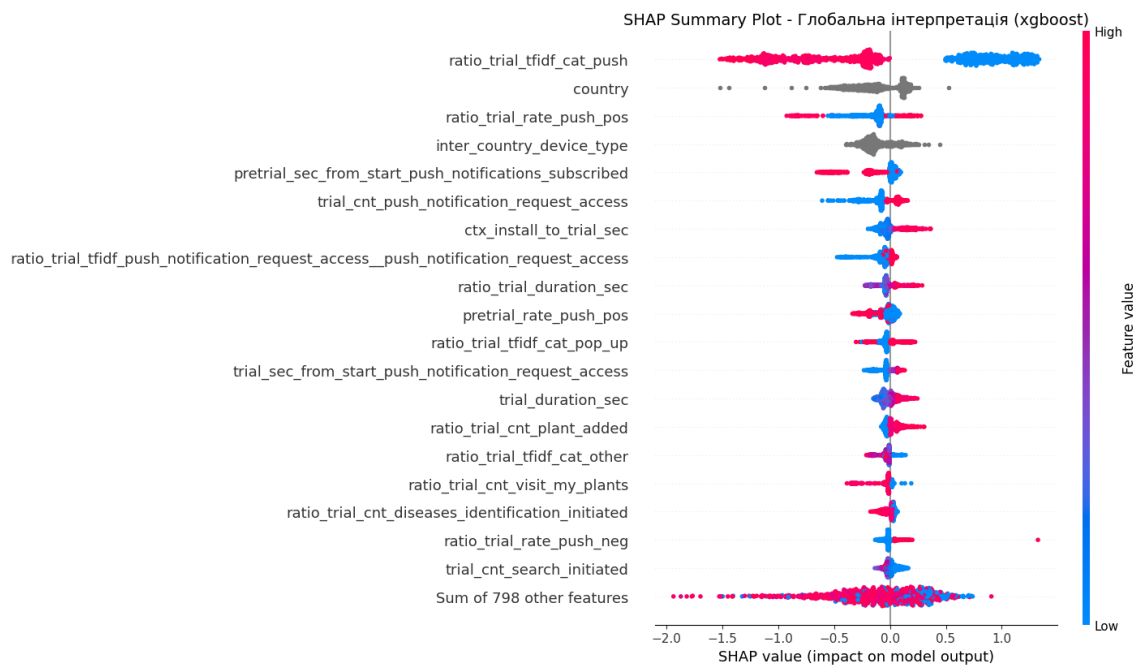


Рисунок 3.10 — Графік SHAP моделі XGBoost

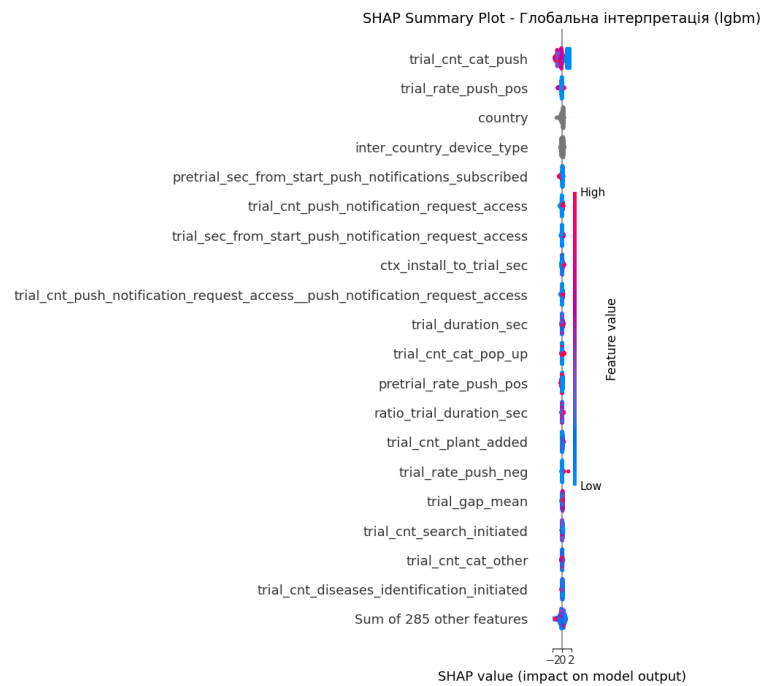


Рисунок 3.11 — Графік SHAP моделі LightGBM

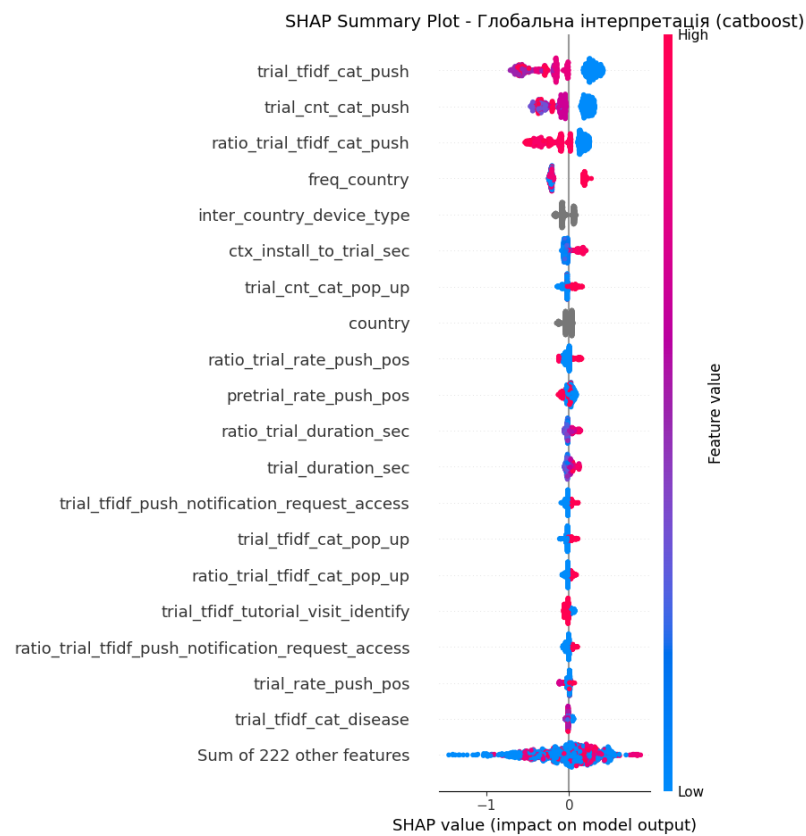


Рисунок 3.12 — Графік SHAP моделі CatBoost

Положення крапки на горизонтальній осі X демонструє величину значення SHAP, тобто маргінальний вплив цієї ознаки на логіт-ймовірність

оформлення підписки, де позитивні значення зсувають прогноз у бік конверсії, негативні — у бік відтоку. Вертикальна вісь Y ранжує ознаки за їхньою сумарною абсолютною предиктивною важливістю. Колірний градієнт крапок від синього до яскраво-червоного кодує фактичне нормалізоване значення самої ознаки у просторі користувача.

Аналіз графіків глобальної інтерпретації (рис. 3.10-3.12) виявляє надзвичайно міцну узгодженість між трьома архітектурами алгоритмів щодо ідентифікації ключових драйверів конверсії. Таблиця 3.6 систематизує найбільш впливові ознаки та їхній вплив на ймовірність успішної бізнес-монетизації.

Таблиця 3.6 — Порівняльний аналіз топових ознак та їхнього впливу на основі SHAP-графіків (XGBoost, LightGBM, CatBoost)

Назва ознаки	Природа та семантика	Напрямок впливу (SHAP)	Верифікація гіпотез (Розділ 2)
ratio_trial_tfidf_cat_push / trial_tfidf_cat_push	TF-IDF взаємодії з пуш-сповіщенням и.	Сильний негативний. Високі значення формують кластери зліва (негативний вплив), низькі — концентруються біля нуля.	Топ-1 ознака у моделях (XGBoost, CatBoost). Підтверджує, що надмірні пуші (спам, тертя в UX) створюють когнітивне навантаження, дратують користувача і гарантовано руйнують монетизацію.

Продовження таблиці 3.6

Назва ознаки	Природа та семантика	Напрямок впливу (SHAP)	Верифікація гіпотез (Розділ 2)
country / freq_country / inter_country_device _type	Демографічні та апаратні індикатори.	Нелінійний / Змішаний. Багатополярні кластери по обидва боки від нуля.	Топ-2/3 позиції. Країна та пристрій задають фундаментальну готовність до оплати. Поведінкові фактори її лише коригують, але не нівелюють.
ratio_trial_rate_push_pos / trial_rate_push_pos	Метрика позитивної взаємодії з пушами.	Негативний. Навіть високі показники "позитивної" взаємодії зміщуються в негативну зону.	Парадокс: велика кількість пушів деструктивна, навіть якщо їх відкривають. Самостійні органічні сесії генерують більшу лояльність, ніж вимушена реакція на часті тригери.
trial_cnt_cat_user_plant / ratio_trial_cnt_plant_added	Метрики залученості до основного функціоналу.	Позитивний. Активне додавання рослин створює сильний позитивний вплив на модель.	Підтверджує важливість органічної взаємодії. Проходження онбордингу, інвестиція часу у власний контент та відчуття цінності продукту є ключовими активаторами конверсії.

Кінець таблиці 3.6

Назва ознаки	Природа та семантика	Напрямок впливу (SHAP)	Верифікація гіпотез (Розділ 2)
pretrial_sec_from_start_push_notifications_subscribed	Час від першої сесії до підписки на сповіщення.	Негативний. Високі значення локалізуються в лівій частині графіка.	Пізній дозвіл на сповіщення свідчить про недовіру чи розгубленість. Зволікання з базовими дозволами статистично знижує ймовірність стати платним клієнтом.

Аналізуючи графіки глобальної інтерпретації, ми також можемо чітко простежити вплив інших специфічних ознак, згаданих у розділі 2, наприклад, `ratio_trial_rate_system_neg` — метрика системної фрустрації. Відповідно до теорії, високий рівень системних помилок та збоїв збільшує когнітивне навантаження. На SHAP-графіках будь-які індикатори переривання користувацького потоку мають виражений негативний вектор, алгоритмічно штрафуючи логіти конверсії. Одночасно, високі показники успішності завдань, такі як `user_tsr`, зосереджені у правій, тобто позитивній, частині графіка. Високий коефіцієнт `user_tsr` сигналізує про відсутність тертя в інтерфейсі, що безпосередньо стимулює довіру до мобільного застосунку.

Таким чином, SHAP-аналіз підтверджує тезу про те, що алгоритми сімейства GBDT не просто завчають статистичний шум, а формують семантичні уявлення про якість користувацького досвіду. Наявність бар'єрів, надмірна ентропія дій та фруструюча комунікація спам сповіщеннями діють як істинні першопричини відмови від монетизації.

У той час як графіки бджолиного рою розкривають макроскопічні тренди когорти, локальна інтерпретація SHAP дозволяє препарувати алгоритм прийняття рішень для кожного індивідуального користувача. Це перетворює

абстрактну модель машинного навчання на практичний інструмент аудиту та гіперперсоналізації, що дозволяє маркетологам розуміти контекст відтоку клієнтів на мікрорівні. Центральним візуальним артефактом для такого аналізу є каскадна діаграма.

Каскадна діаграма SHAP демонструє процес алгебраїчного накопичення доказів. Алгоритм стартує від базового значення $E[f(X)]$ — середнього логіт-прогнозу конверсії по всій тренувальній вибірці. Далі, кожна ознака поточного користувача послідовно додає свій маргінальний вектор SHAP до цієї бази. Сині стрілки вказують на негативні значення, які штовхають прогноз вліво, тобто зменшують шанси на конверсію, тоді як червоні стрілки репрезентують позитивні сили, що зсувають прогноз вправо, збільшують шанси. Фінальна точка агрегації дорівнює фактичному прогнозу моделі $f(x)$ для даного спостереження.

Для детального аналізу були згенеровані каскадні діаграми локальної інтерпретації одного і того ж випадкового користувача з аномально низькою ймовірністю конверсії через призму трьох розроблених моделей, які зображені на рисунках 3.13-3.15.

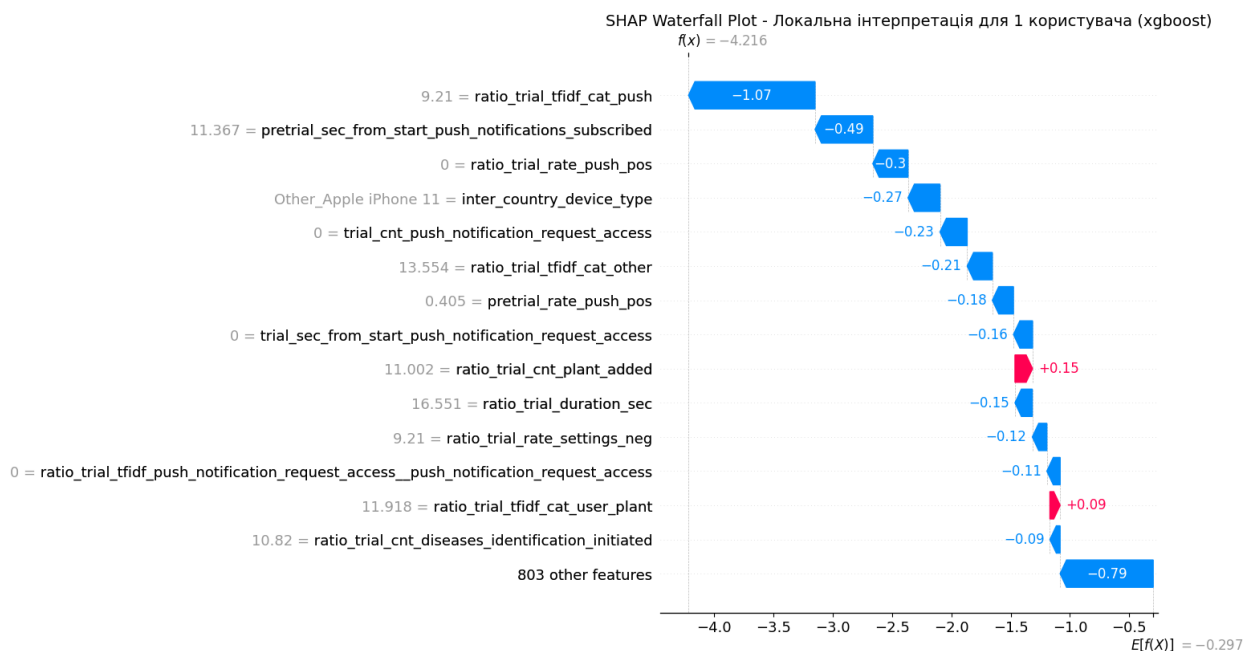


Рисунок 3.13 — Каскадна діаграма локальної інтерпретації одного випадкового користувача моделі XGBoost

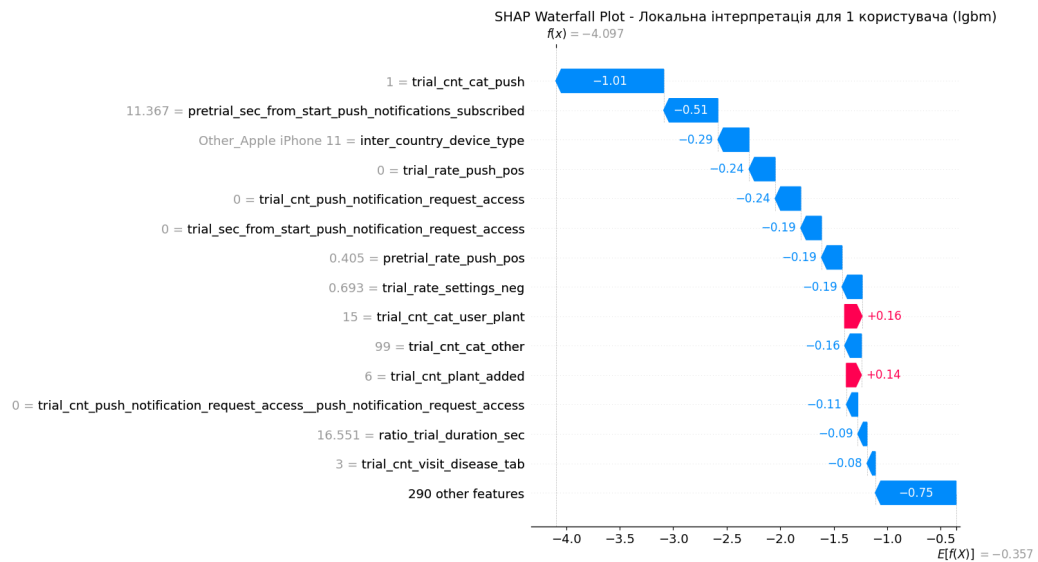


Рисунок 3.14 — Каскадна діаграма локальної інтерпретації одного випадкового користувача моделі LightGBM

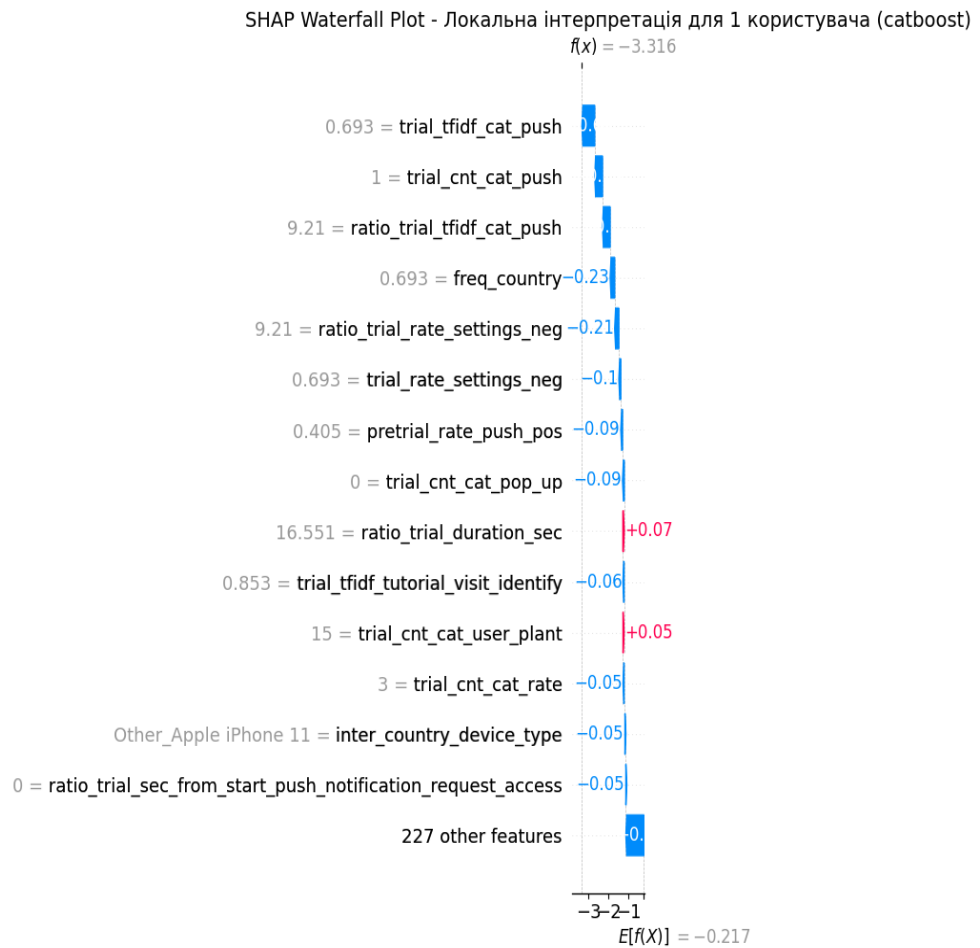


Рисунок 3.15 — Каскадна діаграма локальної інтерпретації одного випадкового користувача моделі CatBoost

Каскадна діаграма XGBoost починається з базового очікуваного значення $E|f(X)| = -0.297$, що відповідає середньому базовому рівню конверсії когорти. Однак, індивідуальний прогноз моделі для цього користувача обвалюється до екстремального значення $f(x) = -4.216$, що означає практично нульову ймовірність оформлення підписки.

Аналіз векторів впливу виявляє, що найбільшим руйнівним фактором стала ознака `ratio_trial_tfidf_cat_push` із фактичним значенням 9.21, яка згенерувала масивний негативний SHAP-вектор -1.07. Наступними за силою негативного впливу виступили часова метрика `pretrial_sec_from_start_push_notifications_subscribed` — значення 11.367, SHAP -0.49, та `ratio_trial_rate_push_pos` — значення 0, SHAP -0.3. Додатково ситуацію погіршили статичні ознаки `inter_country_device_type`.

Єдиними факторами, які зсунули прогноз в позитивний бік, були метрики залученості: `ratio_trial_cnt_plant_added` — значення 11.002, SHAP +0.15, та `ratio_trial_tfidf_cat_user_plant` — значення 11.918, SHAP +0.09.

Як виновок, даний користувач проявляв певну зацікавленість основним функціоналом — додаванням рослин, проте агресивна пуш-комунікація або фруструючі запити на дозволи стали тригером для гарантованого відтоку.

Діаграма асиметричної моделі LightGBM демонструє ідентичний вектор рішень. Від базового значення $E|f(X)| = -0.357$ прогноз стрімко падає до $f(x) = -4.097$. Головним деструктивним фактором знову визнано надлишок пуш-комунікації: ознака `trial_cnt_cat_push` зі значенням 1 згенерувала колосальний штраф -1.01. Значення `pretrial_sec_from_start_push_notifications_subscribed` (-0.51) та апаратний тип пристрою `inter_country_device_type` (-0.29) доповнили негативний сценарій. Як і в XGBoost, позитивні спроби залучення `trial_cnt_cat_user_plant` (+0.16) та `trial_cnt_plant_added` (+0.14) виявилися занадто слабкими, щоб компенсувати негативний користувацький досвід.

Модель CatBoost, що використовує симетричні дерева, стартує з вищої базової точки $E|f(X)| = -0.217$ і завершує прогноз на відмітці $f(x) =$

–3.316. Топологія причин відмови залишається незмінною, хоча ваги розподіляються більш рівномірно завдяки регуляризації. Найбільші негативні SHAP-значення генерують ознаки `trial_tfidf_cat_push` (-0.41), `trial_cnt_cat_push` (-0.31) та `freq_country` (-0.23). Ознака `ratio_trial_duration_sec` зі значенням 16.551 забезпечує незначний позитивний внесок (+0.07).

Синтезуючи результати мікроскопічної локальної інтерпретації по всіх трьох графіках, можна зробити наступний висновок: архітектура пояснювального штучного інтелекту дозволяє беззаперечно довести, що математичні моделі GBDT здатні ізолювати причинно-наслідкові ланцюги відмови від монетизації. Модель діє не лише як предиктивний оракул, але й як діагностичний інструмент, що вказує на конкретні "вузькі місця" в інтерфейсі застосунку. Цей рівень гранулярної прозорості повністю задовольняє бізнес-вимоги до інтеграції систем машинного навчання у виробничий цикл корпоративних продуктів [24].

3.5.2 Калібрування порогу відсічення для Conversions API

Теоретична інтерпретація моделі підтверджує її адекватність, проте генерація математичних ймовірностей конверсії є лише транзитним етапом. Предиктивна система набуває реальної економічної цінності виключно тоді, коли її прогнози операціоналізуються і конвертуються у бізнес-рішення. У сучасній парадигмі перфоманс-маркетингу для мобільних застосунків оптимізація рекламних бюджетів історично спиралася на клієнтські пікселі або SDK, які фіксували детерміновані факти транзакцій.

Для емпіричного визначення оптимальних порогів прийняття рішень було згенеровано графіки калібрування для всіх трьох архітектур — XGBoost (рис. 3.16), LightGBM (рис. 3.17) та CatBoost (рис. 3.18).

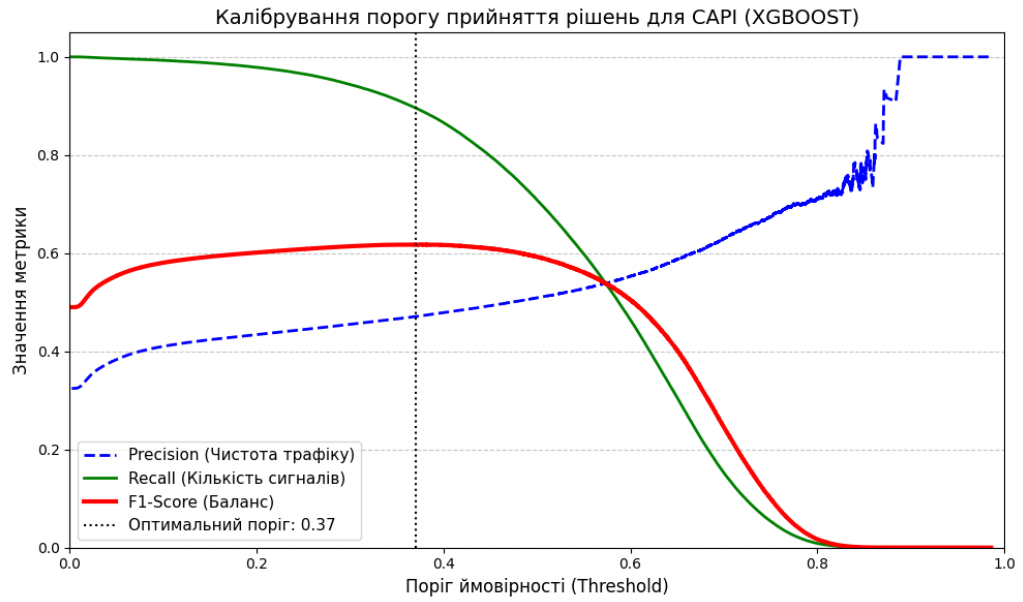


Рисунок 3.16 — Графік калібрування моделі XGBoost

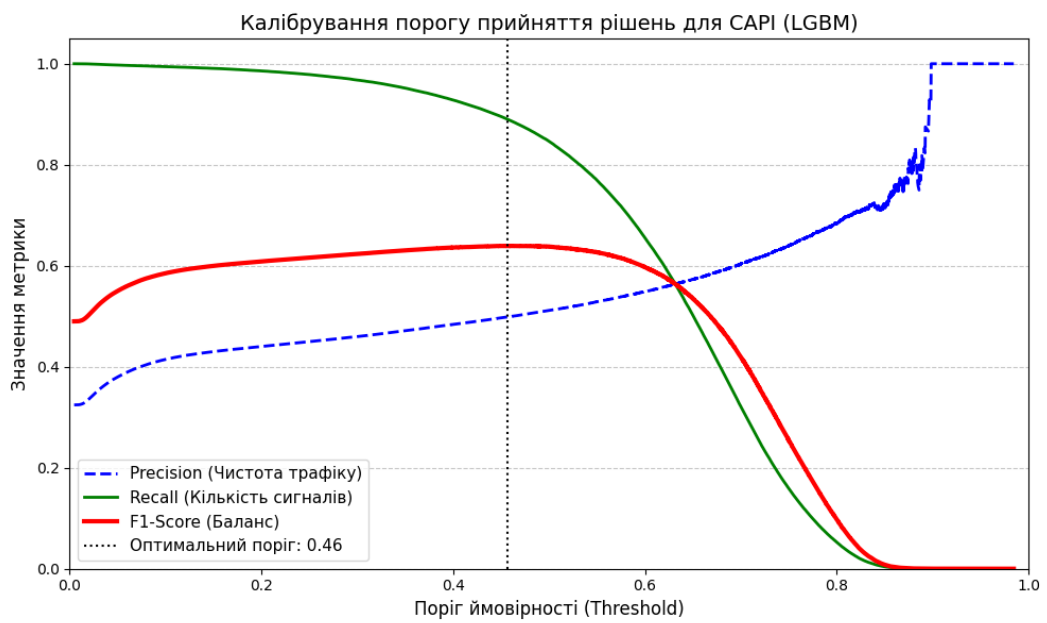


Рисунок 3.17 — Графік калібрування моделі LightGBM

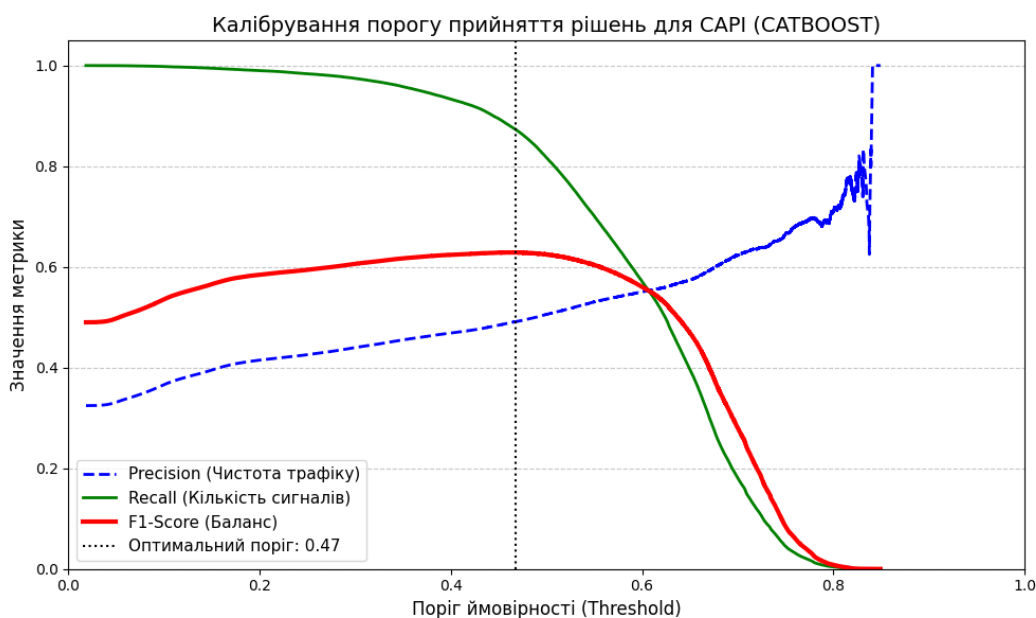


Рисунок 3.18 — Графік калібрування моделі CatBoost

На цих візуалізаціях вісь абсцис позначає спектр можливих порогів ймовірності від 0 до 1. Вісь ординат демонструє значення відповідних класифікаційних метрик. Синя пунктирна лінія репрезентує метрику Precision, зелена суцільна лінія — метрику Recall, а червона лінія візуалізує гармонічне середнє цих двох показників — F1-Score. Вертикальна чорна пунктирна лінія автоматично ідентифікує математичний оптимум.

Дослідження цих кривих виявляє класичний математичний компроміс між Точністю та Повнотою. Чим вищий поріг відсічення, тим стрімкіше падає кількість знайдених конверсій, тобто зелена лінія йде вниз, але одночасно експоненційно зростає впевненість моделі у правильності відібраних сигналів, коли синя лінія йде вгору.

Якщо спиратися виключно на математичну метрику максимізації балансу F1-Score, оптимізатор обчислює наступні точки екстремумів, які позначені на графіках вертикальними чорними пунктирними лініями: Для моделі XGBoost (рис. 3.16) оптимальний математичний поріг становить 0.37, у цій точці криві Precision та Recall перетинаються на рівні близько 0.55; для моделі LightGBM (рис. 3.17) оптимальний математичний поріг зміщується до 0.46, точка перетину кривих Precision та Recall фіксується на рівні 0.62; для

CatBoost (рис. 3.18) оптимальний математичний поріг встановлено на рівні 0.47, перетин кривих спостерігається на позначці близько 0.60.

Проте, як було обґрунтовано у пункті 2.5.2, використання цих теоретичних математичних оптимумів є неприйнятним для бізнес-логіки інтеграції з САРІ, оскільки кожне друге відправлене повідомлення про конверсію буде хибнопозитивним, що неминуче деградує рекламний аукціон [16].

Для того, щоб перетворити предиктивну модель на ефективний інструмент масштабування рекламних кампаній, необхідно радикально змістити поріг відсічення вправо, відмовившись від оптимізації F1-Score на користь максимізації метрики Precision. На основі аналізу кривих на графіках та індустріальних вимог до якості аудиторій, для промислової експлуатації системи обирається поріг ймовірності на рівні 0.75.

Аналізуючи поведінку кривих при досягненні значення осі $X = 0.75$, спостерігаються деградація Повноти та максимізація точності. Зелена крива стрімко падає, опускаючись до рівня приблизно 0.20–0.30. Це означає, що система свідомо відмовляється ідентифікувати близько 70% фактичних майбутніх покупців. З точки зору загальної статистики це виглядає як втрата інформації, проте для алгоритмів Meta Ads щотижневого потоку навіть у 50–100 високоякісних подій абсолютно достатньо для швидкого виходу з фази навчання та стабілізації аукціону. Синя пунктирна крива експоненційно зростає, досягаючи плато на рівні понад 0.85–0.90. Це гарантує, що з усіх згенерованих синтетичних сигналів, які будуть відправлені на сервери рекламної мережі, переважна більшість репрезентуватимуть елітний сегмент користувачів із реальним найвищим наміром до конверсії.

Таблиця 3.7 систематизує результати імітаційного моделювання економічного впливу різних порогів відсічення на ефективність рекламних інвестицій.

Таблиця 3.7 — Імітація бізнес-наслідків при виборі різних порогів відсічення для Conversions API

Поріг	Precision	Recall	Характеристика згенерованого масиву для CAPI	Прогнозований економічний вплив на рекламний бюджет
0.3	Низька (~0.25)	Максимальна (~0.95)	Масив екстремально засмічений. Аномальна кількість хибних спрацювань.	Експоненційне зростання вартості залучення клієнта. Алгоритм Meta оптимізується під нерелевантний трафік. Бюджет витрачається даремно.
0.47	Середня (~0.60)	Середня (~0.60)	Задовільний рівень, проте ризикований. Значна частка сигналів є "шумом".	Стандартна робота аукціону, висока волатильність рентабельності. Якість недостатня для агресивного масштабування на нових ринках.
0.75	Агресивно висока (~0.85+)	Навмисно низька (~0.25)	Високоочищений, преміальний сигнал. Гарантована якість.	Радикальне підвищення ROAS. Рекламний бюджет витрачається виключно на таргетинг аудиторії з найвищим прогнозованим життєвим циклом.

Варто наголосити, що стратегія зсуву порогу до 0.75 має сенс виключно

тоді, коли ймовірнісні прогнози моделі є ідеально відкаліброваними. Якщо алгоритм страждає на гіпер-впевненість, що часто спостерігається у незбалансованих деревах LightGBM, сире значення 0.75 математично не гарантуватиме, що 75 зі 100 таких користувачів реально конвертуються. Оскільки у попередніх розділах було доведено, що симетрична архітектура CatBoost продукує найкращі показники калібрування, застосування порогу 0.75 до її виходів гарантує створення надійного інструменту оптимізації без необхідності додаткового пост-процесингу через ізотонічну регресію [27].

Після прийняття рішення на рівні бекенду про те, що конкретний користувач подолав поріг ймовірності 0.75, система ініціює технологічний процес комунікації з інфраструктурою Meta Graph API.

Для того, щоб рекламна мережа змогла успішно зіставити отриманий синтетичний серверний сигнал із профілем реальної людини у соціальній мережі та використати його для навчання своїх нейромереж, інженерним параметром стає якість збігу подій — Event Match Quality (EMQ). Згідно з технічними регламентами Meta, оцінка EMQ вимірюється за шкалою від 1 до 10, і для розблокування повноцінного потенціалу алгоритмів Advantage+ цей показник повинен стабільно перевищувати відмітку 6.0.

Для забезпечення такого рівня якості, конвеєр збагачення даних повинен додавати до корисного навантаження HTTP-запиту САРІ максимально можливий спектр ідентифікаторів користувача. Усі персональні дані проходять обов'язкову процедуру криптографічного хешування за алгоритмом SHA-256 на стороні сервера мобільного застосунку для забезпечення повної відповідності стандартам конфіденційності (GDPR/CCPA). До структури пакету інтегруються: мережеві індикатори `client_ip_address` та `client_user_agent`, автоматично вилучені з сирих логів найпершої сесії; специфічні рекламні ідентифікатори `fbr` (Facebook browser ID) та `fbc` (Facebook click ID), які генеруються після кліку на рекламу; криптографічно хешовані параметри `em` (електронна пошта) або `ph` (номер телефону), за умови, що користувач надав їх під час реєстраційних процесів в застосунку.

Оскільки в перехідний період застосунок може продовжувати використовувати традиційний клієнтський трекінг паралельно з серверним відправленням синтетичних подій через САРІ, у мікросервіс інтегровано алгоритм дедуплікації. Кожній згенерованій події присвоюється унікальний ідентифікатор `event_id`. Якщо рекламна система Meta фіксує ідентичні `event_id`, що надійшли різними каналами протягом 48 годин, вона автоматично ігнорує дублікати. Ця архітектурна запобіжна міра унеможливорює штучне роздуття показників конверсій та хибну оптимізацію витрат у рекламних кабінетах маркетологів [5, 16].

3.6 Висновки до розділу 3

У третьому розділі було здійснено комплексну програмну реалізацію та імітацію бізнес-інтеграції системи предиктивної аналітики для прогнозування конверсії користувачів мобільного застосунку.

По-перше, спроектовано високонавантажену архітектуру конвеєра обробки даних на базі Python 3.12 та фреймворку лінійних обчислень Polars, що дозволило ефективно опрацьовувати багатогігабайтні масиви сирих логів в умовах апаратних обмежень. Розроблений конвеєр забезпечує суворий захист від витоків цільової змінної через механізм хронологічного усічення сесій та реалізує екстракцію ознак. Завдяки впровадженню багаторівневої таксономії подій було алгоритмізовано складні метрики користувацького досвіду, такі як Task Success Rate, Cognitive Effort Score та ентропію дій. Крім того, секвенціальна природа поведінки була успішно трансформована у розріджені матриці за допомогою алгоритмів TF-IDF векторизації n-грамів.

По-друге, розроблено алгоритм раннього злиття гетерогенних просторів ознак. Для стабілізації дисперсії гіпервимірному простору та боротьби з "прокляттям розмірності" успішно імплементовано методи просторової

нормалізації, а саме відсікання аномалій P99 Clipping та зсунута логарифмізація Log1p, та дисперсійну фільтрацію квазіконстантних ознак. Це дозволило сформуванню репрезентативний і стійкий до перенавчання датасет.

По-третє, проведено хронологічно ізольований порівняльний аналіз трьох алгоритмів градієнтного бустингу: XGBoost, LightGBM та CatBoost. Хоча LightGBM продемонстрував найвищу повноту виявлення, а XGBoost — найкраще калібрування ймовірностей, фінальною моделлю для промислової експлуатації було обрано CatBoost. Архітектура його симетричних дерев забезпечила найоптимальніший баланс дискримінаційних метрик, стійкість до категоріальних ознак без потреби у пам'яті-ємному One-Hot кодуванні та безпрецедентну швидкість інференсу, що є важливим критерієм для інтеграції в мікросервісний бекенд.

По-четверте, застосування методів пояснювального штучного інтелекту дозволило здійснити глобальну та локальну деконструкцію алгоритмічних прогнозів. Аналіз візуалізацій, зокрема графіків бджолиного рою та каскадних діаграм, математично довів, що модель приймає рішення на основі об'єктивної якості користувацького досвіду. Було верифіковано теоретичні гіпотези: надмірне когнітивне навантаження діє як найпотужніший драйвер відтоку, тоді як органічна взаємодія з основним функціоналом стимулює конверсію.

Нарешті, було обґрунтовано архітектуру бізнес-інтеграції предиктивної моделі з рекламними платформами через протокол API для подолання проблеми втрати клієнтських cookie та обмежень App Tracking Transparency. Аналіз графіків калібрування довів необхідність відмови від стандартного порогу класифікації 0.5 на користь асиметричного порогу 0.75. Таке екстремальне зміщення дозволяє максимізувати Точність, генеруючи високоочищений потік "синтетичних конверсій". Відправка цих преміальних сигналів у рекламну мережу гарантує навчання алгоритмів виключно на користувачах із найвищим наміром до покупки, запобігаючи деградації аукціону та забезпечуючи стаке зростання рентабельності рекламних інвестицій.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ АПАРАТНО-ПРОГРАМНОГО ЗАСОБУ ТА ПРОГРАМНОГО ПРОДУКТУ

Перехід ринку мобільних застосунків до моделі підписок та зростання вартості залучення клієнтів вимагають швидкого оцінювання окупності користувачів. Оскільки традиційна аналітика занадто повільна, виникає гостра потреба у предиктивних ML-системах, здатних прогнозувати конверсію в реальному часі на основі поведінки під час тріалу.

Розробка таких систем є ресурсомісткою, тому для максимізації рентабельності інвестицій та уникнення надлишкового ускладнення архітектури застосовується функціонально-вартісний аналіз (ФВА). Цей метод декомпозує продукт на функції та дозволяє знайти найефективніше співвідношення між технічною якістю та витратами на розробку [31].

4.1 Постановка задачі проектування

Об'єктом даного техніко-економічного дослідження є архітектура та алгоритмічне забезпечення системи прогнозування конверсії користувачів мобільного застосунку. Головною метою створення цього програмного продукту є підвищення ефективності процесів залучення та монетизації аудиторії шляхом раннього виявлення користувачів із високою ймовірністю переходу у платну підписку. Замість того, щоб очікувати завершення ознайомчого періоду для фіксації факту оплати, система повинна аналізувати гетерогенні поведінкові логи користувача у перші години його взаємодії з інтерфейсом і генерувати прогнозну ймовірність його подальшої поведінки. Якщо ця ймовірність перетинає певний математично відкалібрований поріг, система ідентифікує користувача як платоспроможного і негайно відправляє

синтетичний сигнал конверсії до рекламних платформ. Такий підхід дозволяє алгоритмам таргетингу рекламних мереж прискорено навчатися та оптимізувати закупівлю трафіку, максимізуючи прибуток бізнесу.

Для виконання цієї задачі програмний комплекс повинен задовольняти низку технічних, функціональних та економічних вимог. Специфіка обробки мобільної телеметрії полягає у безперервному надходженні великих обсягів неструктурованих даних, що створює екстремальне навантаження на підсистеми оперативної пам'яті та центрального процесора. Відповідно, вимоги до системи класифікуються наступним чином:

1. Масштабованість та стійкість до перевантажень, адже з огляду на те, що обсяг вхідних логів вимірюється десятками гігабайтів, система повинна мати архітектуру, здатну обробляти дані потоково або батчами. Використання традиційних аналітичних інструментів, що вимагають повного завантаження датасету в оперативну пам'ять, є неприпустимим через ризик падіння серверів та експоненційного зростання витрат на оренду хмарних ресурсів.
2. Адаптивна екстракція ознак, тобто продукт повинен автоматично конструювати багаторівневу таксономію подій, перетворюючи сирі дискретні логи на неперервні метрики фрустрації та когнітивного навантаження користувача. Крім того, вимагається імплементація методів обробки природної мови, зокрема алгоритму TF-IDF, для векторизації хронологічних послідовностей дій користувача в інтерфейсі, що створює високорозмірний і надзвичайно розріджений простір ознак.
3. Висока точність в умовах дисбалансу класів, адже оскільки конверсія є статистично рідкісною подією на фоні масового відтоку користувачів, класичні алгоритми машинного навчання мають тенденцію до упередженості на користь мажоритарного класу. Обраний алгоритм бінарної класифікації повинен містити внутрішні механізми балансування, щоб максимізувати визначені метрики для контролю

якості моделі.

4. Низька затримка інференсу, тобто процес проходження вектора ознак нового користувача через навчений ансамбль дерев рішень не повинен перевищувати кількох десятків мілісекунд.
5. Інтерпретованість та прозорість рішень, тому що бізнес-аналітики та маркетологи повинні мати інструментарій для розуміння маргінального впливу кожної ознаки на фінальний прогноз, що вимагає сумісності обраного алгоритму з математичними методами пояснювального штучного інтелекту, такими як ігрові значення Шеплі.
6. Економічна доцільність, а саме — сумарні витрати на проектування, написання коду, оптимізацію гіперпараметрів, тренування моделей та подальшу експлуатацію інфраструктури повинні бути мінімізовані. Рішення, які вимагають надмірних трудовитрат або неефективно утилізують апаратні ресурси, мають бути відхилені на користь більш сучасних та оптимізованих технологій.

Кожна з наведених вимог накладає обмеження на простір можливих рішень. Використання апарату функціонально-вартісного аналізу дозволить декомпонувати цей комплексний виклик, проаналізувати кожну підсистему крізь призму її вартості та технічної ефективності, і синтезувати архітектуру, яка найкращим чином задовольнятиме потреби бізнесу.

4.2 Обґрунтування функцій програмного продукту

Відповідно до принципів функціонально-вартісного аналізу, дослідження розпочинається з чіткого формулювання головної функції об'єкта, яка відображає його ключове призначення. Для системи, що розглядається, головна функція F_0 — створення програмної системи прогнозування ймовірності конверсії користувачів мобільного застосунку на

основі гетерогенних поведінкових патернів в умовах обмежених апаратних ресурсів.

Зважаючи на складну багаторівневу архітектуру сучасних конвеєрів машинного навчання, головну функцію F_0 необхідно декомпонувати на логічно завершені підфункції першого рівня, які безпосередньо впливають як на технічні характеристики системи, так і на економічні показники розробки та експлуатації. У результаті системного аналізу виокремлено чотири підфункції.

F_1 — реалізація конвеєра обробки даних. Відповідає за швидку агрегацію та фільтрацію сирих гетерогенних логів із жорстким контролем і оптимізацією споживання оперативної пам'яті.

F_2 — векторизація поведінкових послідовностей. Здійснює розрахунок метрик фрустрації та трансформацію хронологічних дій користувача у числові ознаки за допомогою NLP-алгоритмів.

F_3 — імплементація алгоритму класифікації. Забезпечує навчання на історичних даних та генерацію ймовірнісних прогнозів конверсії з використанням ансамблевих алгоритмів градієнтного бустингу.

F_4 — інтерпретація та бізнес-інтеграція. Визначає вплив окремих ознак на прогноз та калібрує пороги ймовірності для оптимальної передачі сигналів у маркетингові системи.

Для кожної з визначених підфункцій існує кілька альтернативних варіантів технологічної реалізації.

Функція F_1 :

- 1) використання традиційної бібліотеки Pandas;
- 2) використання фреймворку Polars.

Функція F_2 :

- 1) базова агрегація — підрахунок абсолютних частот подій, ONE-кодування;
- 2) застосування методів NLP: генерація n-грам та зважування за алгоритмом TF-IDF із подальшим відсіканням.

Функція F_3 :

- 1) модель XGBoost;
- 2) модель LightGBM;
- 3) модель CatBoost.

Функція F_4 :

- 1) використання стандартного Feature Importance та базового порогу ймовірності;
- 2) глобальна/локальна інтерпретація через SHAP-аналіз та динамічне калібрування порогу для максимізації F1-Score.

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

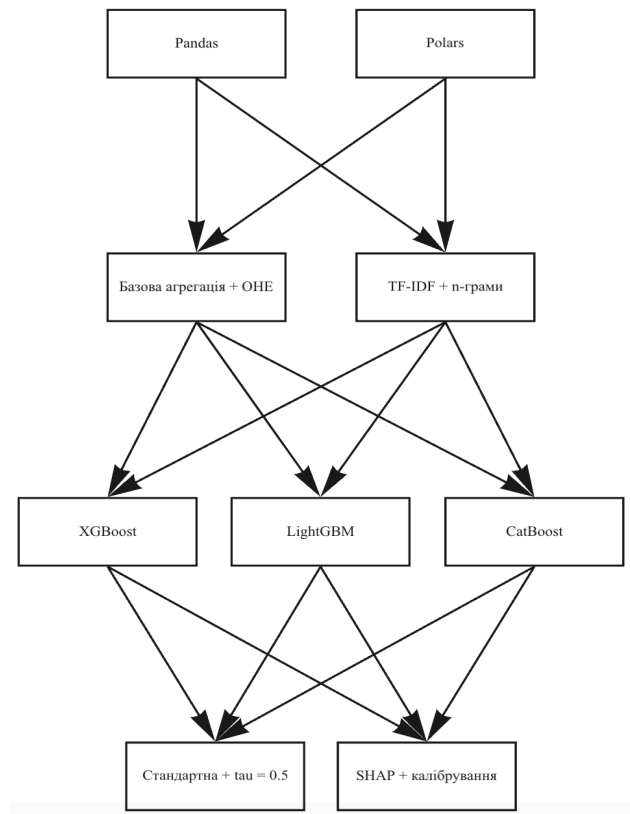


Рисунок 4.1 — Морфологічна карта

За допомогою морфологічної карти є можливість побудувати позитивно-негативну матрицю з відображенням усіх варіантів основних функцій (табл. 4.1).

Таблиця 4.1 — Позитивно-негативна матриця архітектурних рішень

Основна функція	Варіант реалізації	Переваги	Недоліки
F1	1	Простий синтаксис.	Переповнення RAM, однопотоковість.
	2	Економія RAM, багатопотоковість.	Складніша реалізація та синтаксис запитів.
F2	1	Простота, висока швидкість обчислень.	Втрата контексту, експоненційне зростання розмірності.
	2	Збереження хронології, відсів статистичного шуму.	Складність обробки, генерація розріджених матриць.
F3	1	Точність (апроксимація 2-го порядку).	Високе споживання RAM.
	2	Найвища швидкість навчання.	Ризик перенавчання через ріст по листках.
	3	Нативна обробка категорій, швидкий інференс.	Більша ресурсоемність самого процесу навчання.
F4	1	Відсутність додаткових обчислювальних витрат.	Математичне зміщення, ігнорування дисбалансу класів.
	2	Прозорість, максимізація точності.	Витрати процесорного часу на генерацію значень.

Для функції F_1 варіант 1) Pandas є технічно неприйнятним, зважаючи на те, що система повинна конкатенувати масивні розріджені структури зі щільними демографічними матрицями, жадібна природа Pandas призведе до

гарантованого переповнення фізично доступних обсягів оперативної пам'яті на стандартних серверах. Перехід на важкі хмарні інстанси з надлишковим обсягом RAM є економічно нерентабельним. Тому вибором стає архітектура на базі 2) Polars, яка завдяки колонковому формату та лінівим обчисленням математично гарантує виконання потокових операцій у межах доступної пам'яті.

Для функції F_2 варіант 1) Базова агрегація + ONE відкидається з алгоритмічних міркувань. Використання One-Hot Encoding для тисяч висококардинальних поведінкових подій генерує експоненційне зростання розмірності матриці та призводить до повної втрати хронологічного контексту сесії (порядку дій користувача). Лише варіант 2) застосування методів NLP здатне зберегти секвенціальну природу людської поведінки та алгоритмічно відсіяти статистичний шум.

Для функції F_4 варіант 1) Стандартна інтерпретація з порогом визнається нежиттєздатним. В умовах екстремального дисбалансу класів мобільної аналітики використання базового порогу призведе до математичного зміщення та ігнорування алгоритмом міноритарного класу. Отже, необхідним є застосування варіанта 2) SHAP-аналіз та динамічне калібрування порогу, що забезпечує прозорість та максимізацію ключової бізнес-метрики.

Таким чином, простір альтернатив звужується виключно до вибору алгоритму машинного навчання — функції F_3 . Алгоритм XGBoost виключається з подальшого розгляду через його архітектурну особливість генерувати внутрішню структуру DMatrix, що призводить до майже подвійного споживання оперативної пам'яті та знижує стабільність системи під час навчання на високорозмірних TF-IDF матрицях у порівнянні з конкурентами.

У результаті ФВА формуються два варіанти програмного продукту, які підлягають подальшому глибокому кількісному та економічному порівнянню:

- 1) $F_1(2) - F_2(2) - F_3(2) - F_4(2)$;
- 2) $F_1(2) - F_2(2) - F_3(3) - F_4(2)$.

4.3 Обґрунтування системи параметрів програмного продукту

Для переходу від якісних міркувань до точних математичних розрахунків необхідно сформулювати систему техніко-економічних параметрів. Ця система повинна вичерпно характеризувати як продуктивність майбутнього програмного продукту в умовах промислової експлуатації, так і рівень витрат ресурсів на його створення. На основі специфіки Machine Learning Operations та вимог до високонавантажених мікросервісів було виокремлено п'ять параметрів.

X_1 — Швидкість інференсу. Цей параметр є пріоритетом з технічної точки зору. Оскільки система працює як синхронний проксі-сервер між мобільним застосунком та САРІ, час обчислення ймовірності конверсії має бути мінімальним. Висока затримка призведе до накопичення черг у балансувальниках навантаження, таймаутів з'єднань та, як наслідок, втрати цінних маркетингових сигналів. Чим менший цей показник, тим вища якість системи.

X_2 — Транзитне споживання оперативної пам'яті під час навчання. Навчання моделей на датасетах, що містять сотні тисяч векторів із високою розмірністю, є надзвичайно вимогливим до RAM. Параметр відображає максимальний пік, необхідний фреймворку для побудови дерев. Економічний вплив цього параметра прямий: моделі з низьким споживанням пам'яті можуть тренуватися на дешевших хмарних інстансах загального призначення, тоді як високе споживання вимагає оренди дорогих Memory-Optimized серверів.

X_3 — Час навчання моделі. Характеризує швидкість проходження алгоритмом градієнтного спуску та побудови ансамблю. В умовах мобільної аналітики поведінкові патерни користувачів швидко еволюціонують, що вимагає періодичного донавчання моделі. Короткий час навчання значно економить машинний час та прискорює цикл досліджень для Data Science

команди.

X_4 — Точність прогнозування. Цей параметр є головною бізнес-метрикою якості класифікації в умовах дисбалансу класів. Високий показник точності гарантує, що через CAPІ до рекламних мереж надходитимуть виключно релевантні дані, що безпосередньо запобігає неефективному витрачання маркетингових бюджетів на хибні спрацювання та одночасно максимізує охоплення реальної платоспроможної аудиторії. Чим вищий цей показник, тим більша комерційна цінність розробленої системи.

X_5 — Бізнес-інтерпретованість та прозорість. Цей параметр визначає здатність моделі не лише видавати кінцеву ймовірність, але й аргументовано пояснювати логіку прийняття кожного рішення. Оскільки результати прогнозування є основою для розподілу бюджетів та змін у UX/UI дизайні, використання закритих моделей несе значні управлінські ризики.

Спираючись на результати емпіричних лабораторних бенчмарків, проведених на відкладених вибірках, для кожного з визначених параметрів було встановлено граничні значення: найгірше, середнє та найкраще. Ці значення формують математичний базис для подальшого переведення фізичних величин у безрозмірну бальну шкалу оцінювання. Зведена інформація представлена у таблиці 4.2.

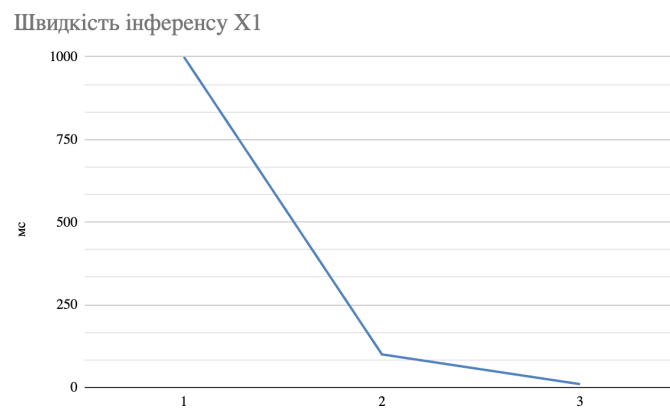
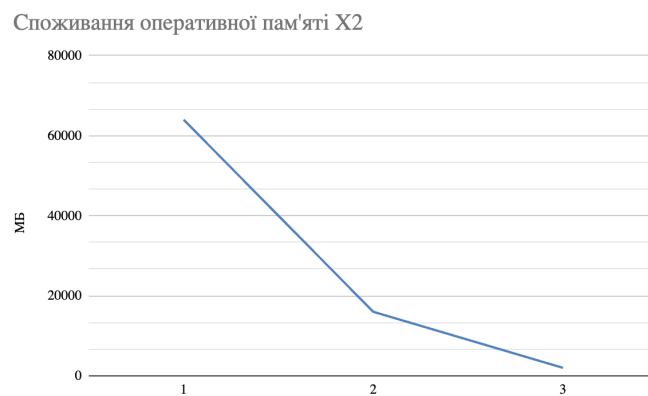
Таблиця 4.2 — Основні параметри програмного продукту та їх граничні значення

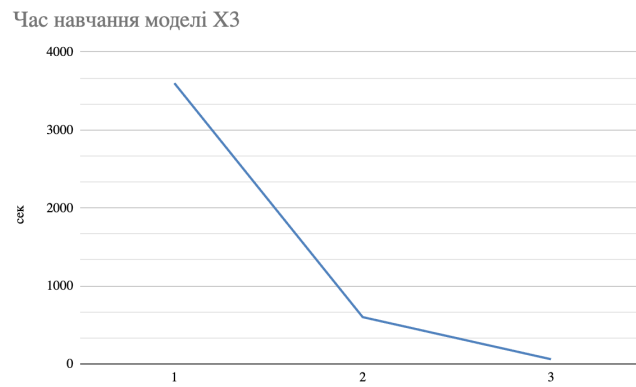
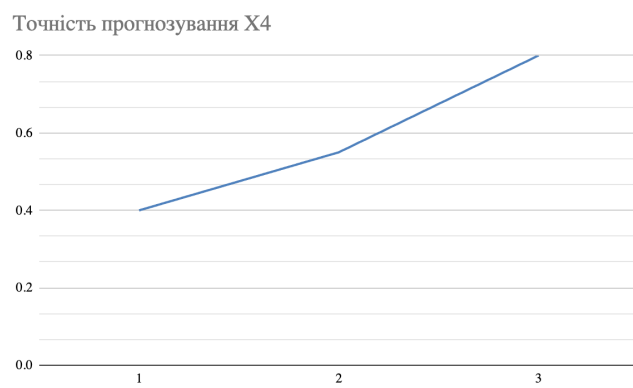
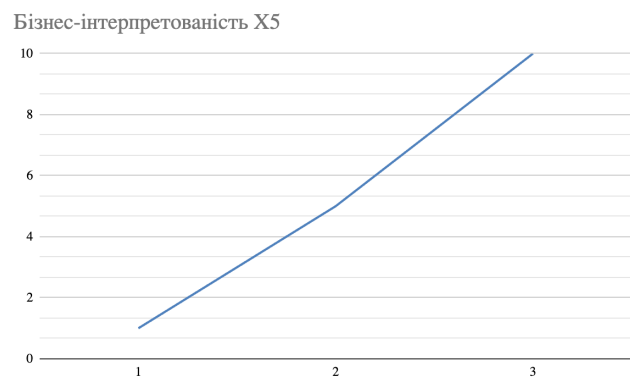
Назва параметра	Умовне позначення	Одиниці виміру	Найгірше значення	Середнє значення	Найкраще значення
Швидкість інференсу	X1	мс	1000	100	10
Споживання оперативної пам'яті	X2	МБ	64000	16000	2000
Час навчання моделі	X3	сек	3600	600	60

Продовження таблиці 4.2

Назва параметра	Умовне позначення	Одиниці виміру	Найгірше значення	Середнє значення	Найкраще значення
Точність прогнозування	X4	Частка одиниці	0.4	0.55	0.8
Бізнес-інтерпретованість	X5	бали	1	5	10

За даними таблиці 4.2 будуються графічні характеристики параметрів, як показано на рис. 4.2-4.6.

Рисунок 4.2 — X_1 , швидкість інференсуРисунок 4.3 — X_2 , споживання оперативної пам'яті

Рисунок 4.4 — X_3 , час навчанняРисунок 4.5 — X_4 , точність прогнозуванняРисунок 4.6 — X_5 , бізнес-інтерпритованість

4.4 Аналіз експертного оцінювання параметрів

Визначення об'єктивної вагомості кожного з параметрів у загальній структурі якості програмного продукту є важливим етапом ФВА. Оскільки технічні та економічні показники мають різну розмірність та вплив на фінальну ефективність, їхня вага визначається за допомогою експертних методів оцінювання.

Для забезпечення високої репрезентативності та уникнення суб'єктивного зміщення до процесу оцінювання було залучено експертну комісію у складі 7 кваліфікованих фахівців. Процедура проходила у кілька етапів. На першому етапі кожен експерт незалежно здійснив ранжування параметрів, присвоївши їм ранги від 1 до 4, де ранг 1 відповідає найважливішому параметру. Результати експертного ранжування зведено у таблицю 4.3.

Таблиця 4.3 — Результати ранжування параметрів експертною комісією

Позначення параметра	Назва параметра	Одиниці вимірювання	Ранг параметра за оцінкою експерта							Сума рангів, R_i	Відхилення, Δ_i	Квадрат відхилення, Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкість інференсу	мс	2	2	1	3	1	2	2	13	-8	64
X2	Споживання оперативної пам'яті	МБ	4	3	4	5	4	5	5	30	9	81

Продовження таблиці 4.3

Позначення параметра	Назва параметра	Одиниці вимірювання	Ранг параметра за оцінкою експерта							Сума рангів, R_i	Відхилення, Δ_i	Квадрат відхилення, Δ_i^2
			1	2	3	4	5	6	7			
X3	Час навчання моделі	сек	5	5	5	4	5	4	4	32	11	121
X4	Точність прогнозування	Частка одиниці	1	1	2	1	2	1	1	9	-12	144
X5	Бізнес-інтерпретованість	бали	3	4	3	2	3	3	3	21	0	0
	Усього		15	15	15	15	15	15	15	105	0	410

Щоб оцінити достовірність експертних рейтингів, визначаємо такі параметри.

1. Сумарний ранг кожного показника та загальний ранг усіх показників $N \times n$:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{N \cdot n \cdot (n+1)}{2} = \frac{7 \cdot 5 \cdot 6}{2} = 105,$$

де $N = 7$ експертів,

$n = 5$ параметри.

Розрахована сума у таблиці збігається з теоретичною, що виключає арифметичні помилки.

2. Визначення середньої суми рангів:

$$T = \frac{1}{n} R_i = 21.$$

3. Обчислення індивідуальних відхилень суми рангів кожного параметра від середньої:

$$\Delta_i = R_i - T.$$

$-8 + 9 + 11 - 12 + 0 = 0$. Умова виконується.

4. Розрахунок загальної суми квадратів відхилень:

$$S = \sum_{i=1}^N \Delta_i^2 = 64 + 81 + 121 + 144 + 0 = 410.$$

5. Обчислення коефіцієнта конкордації за стандартизованою формулою:

$$W = \frac{12 \cdot S}{N^2 \cdot (n^3 - n)} = \frac{12 \cdot 410}{49 \cdot 120} = \frac{41}{49} \approx 0.837.$$

Згідно з нормативними вимогами для радіотехнічних виробів та програмних комплексів, експертиза вважається надійною, якщо розрахункове значення W перевищує порогове значення $W_{\text{норм}} = 0.67$. Оскільки отримане значення $0.837 > W_{\text{норм}} = 0.67$, зібрані оцінки визнаються достовірними та придатними для подальшого застосування. Таким чином утворимо таблицю 4.4, де попарно порівняємо всі параметри.

Таблиця 4.4 — Попарне порівняння параметрів.

[illegible]

На наступному етапі експерти здійснюють попарне порівняння всіх параметрів. Метод попарного порівняння дозволяє усунути когнітивні спотворення при оцінці багатовимірних об'єктів. Використовується класична шкала переваг: якщо параметр X_i має більший пріоритет за X_j , експерт виставляє бал 1.5; якщо менший пріоритет — 0.5; у випадку рівнозначності — 1.0.

Після агрегації думок усіх експертів формується матриця переваг $A = \|a_{ij}\|$, на основі якої розраховуються відносні вагові коефіцієнти. Для кожного параметра розрахунок первинної вагомості K_{bi} здійснюється за наступними формулами:

$$K_{bi} = \frac{b_i}{\sum_{j=1}^n b_j}, \quad (4.2)$$

$$b_i = \sum_{j=1}^n a_{ij}. \quad (4.3)$$

Для досягнення математичної стабільності результатів алгоритм вимагає проведення ітераційних розрахунків. Відносні оцінки розраховуються декілька разів. На другому і наступних кроках відносні оцінки перераховуються з урахуванням значень попередньої ітерації за такими формулами:

$$K_{bi} = \frac{b'_i}{\sum_{j=1}^n b'_j}, \quad (4.4)$$

$$b'_i = \sum_{j=1}^n a_{ij} b_j. \quad (4.5)$$

Ітерації тривають доти, поки наступні значення не будуть незначно відрізнятися від попередніх, тобто поки різниця значень коефіцієнтів вагомості між сусідніми кроками не стане меншою за 5%. Результати обчислень зведено у таблицю 4.5.

Таблиця 4.5 — Ітераційний розрахунок вагомості параметрів

Параметри	Параметри x_j					Перша ітерація		Друга ітерація	
x_i	X1	X2	X3	X4	X5	b_i	K_{bi}	b_i^1	K_{bi}^1
X1	1	1.5	1.5	0.5	1.5	6	0.24	27.5	239
X2	0.5	1	1.5	0.5	0.5	4	0.16	17.5	152
X3	0.5	0.5	1	0.5	0.5	3	0.12	14	122
X4	1.5	1.5	1.5	1	1.5	7	0.28	34	296
X5	0.5	1.5	1.5	0.5	1	5	0,20	22	191
Всього:						25	1	115	1

Порівняння значень K_{bi} та K_{bi}^1 свідчить про те, що розбіжність є мінімальною, отже, процес ітераційного розрахунку успішно зійшовся на другому кроці. Фінальні вагові коефіцієнти підтверджують гіпотезу про те, що для мікросервісу предиктивної аналітики ключовим показником є точність прогнозування, оскільки саме від неї залежить якість маркетингових сигналів та ефективність витрачання бюджетів. Друге місце в ієрархії пріоритетів обґрунтовано посідає швидкість інференсу, адже система має працювати як високопродуктивний проксі-сервер і миттєво віддавати результати через САРІ без затримок і таймаутів. Натомість час навчання моделі та споживання оперативної пам'яті отримали найменший пріоритет, оскільки процеси тренування та побудови дерев виконуються у фоновому режимі, а сучасна хмарна архітектура дозволяє відносно легко масштабувати апаратні ресурси за потребою.

4.5 Аналіз рівня якості варіантів реалізації функцій

Синтезувавши експертні вагові коефіцієнти та об'єктивні технічні характеристики алгоритмів, переходимо до розрахунку інтегрального узагальнюючого показника якості для досліджуваних варіантів архітектури. Фізичні абсолютні значення параметрів для алгоритмів були отримані під час обчислювальних бенчмарків. За допомогою лінійної інтерполяції ці значення трансформуються у бальну оцінку B_{ij} за 10-бальною шкалою.

Характеристики Варіанту 1 (Polars + LightGBM + CAPI + SHAP):

1. X_1 (Швидкість інференсу) становить 56.21 мс. Цей показник кращий за середню границю, проте в кілька разів поступається ідеалу. Асиметричний ріст дерев LightGBM призводить до непередбачуваних проходів по графу, руйнуючи просторову локальність кешу процесора під час інференсу. Оцінка: 7 балів.
2. X_2 (Споживання пам'яті) становить 142.83 МБ. Завдяки механізму об'єднання взаємовиключних розріджених ознак, алгоритм екстремально ощадливий до транзитної RAM, значно перевищуючи найкраще граничне значення. Оцінка: 10 балів.
3. X_3 (Час навчання) становить 412.16 сек. Модель навчається на масиві TF-IDF із середньою швидкістю. Оцінка: 6 балів.
4. X_4 (Точність прогнозування) становить 0.5132. В умовах екстремального дисбалансу класів модель демонструє метрику, що наближається до середнього цільового порогу. Оцінка: 4 бали.
5. X_5 (Бізнес-інтерпретованість) інтеграція алгоритму SHAP забезпечує високу прозорість прийняття рішень. Проте асиметрична структура дерев LightGBM дещо ускладнює математичний розрахунок ігрових значень Шеплі порівняно із симетричними аналогами. Оцінка: 8 балів.

Характеристики Варіанту 2 (Polars + CatBoost + CAPI + SHAP):

1. X_1 (Швидкість інференсу) становить 5.76 мс. Завдяки топології

симетричних дерев інференс зводиться до елементарної побітової операції в кеші CPU. Показник фактично ідеальний і перевершує найкращу границю. Оцінка: 10 балів.

2. X_2 (Споживання пам'яті) становить 1009.51 МБ. Динамічний розрахунок впорядкованих цільових статистик на множинних перестановках вибірки вимагає більших транзитних буферів у RAM. Проте показник все ще вдвічі кращий за найкращу границю. Оцінка: 10 балів.
3. X_3 (Час навчання) становить 137.36 сек. Модель сходиться дуже швидко, наближаючись до ідеального показника. Оцінка: 9 балів.
4. X_4 (Точність прогнозування) становить 0.5056. Показник чистоти маркетингового сигналу дещо поступається LightGBM, знаходячись у робочому діапазоні для незбалансованих логів. Оцінка: 4 балів.
5. X_5 (Бізнес-інтерпретованість) симетрична геометрія дерев CatBoost ідеально поєднується з математичним апаратом TreeExplainer від SHAP. Це гарантує бездоганну швидкість розрахунку локальних та глобальних пояснень прогнозу без математичних похибок. Оцінка: 10 балів.

Узагальнюючий показник технічного рівня якості розраховується як сума добутків вагових коефіцієнтів на відповідні бальні оцінки (табл. 4.6):

$$K_k(j) = \sum_{i=1}^n K_{vi,j} \cdot B_{ij}, \quad (4.6)$$

де n — кількість параметрів;

K_{vi} — коефіцієнт вагомості i -го параметра;

B_i — оцінка i -го параметра в балах.

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

Варіант реалізації	Параметри	Абсолютне значення параметра	Бальна оцінка параметра. B_i	Коефіцієнт вагомості параметра. K_{Bi}	Коефіцієнт рівня якості $(B_i K_{Bi})$
Варіант 1 (LightGBM)	X1	56.21	7	0.239	1.673
	X2	142.83	10	0.152	1.520
	X3	412.16	6	0.122	0.732
	X4	5.132	4	0.296	1.184
	X5	-	8	0.191	1.528
Варіант 2 (CatBoost)	X1	5.76	10	0.239	2.390
	X2	1.009.51	10	0.152	1.520
	X3	137.36	9	0.122	1.098
	X4	5.056	4	0.296	1.184
	X5	-	10	0.191	1.910

За даними з таблиці 4.6 за формулою:

$$K_k = K_{Ty}[F_{1k}] + K_{Ty}[F_{2k}] + \dots + K_{Ty}[F_{zk}]. \quad (4.7)$$

визначаємо рівень якості кожного з варіантів:

$$K_{k1} = 1.673 + 1.520 + 0.732 + 1.184 + 1.528 = 6.637$$

$$K_{k2} = 2.390 + 1.520 + 1.098 + 1.184 + 1.910 = 8.102$$

Як видно з розрахунків, кращим є Варіант 2, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки програмного продукту (ПП) спочатку проведемо розрахунок його трудомісткості. Обидва варіанти включають в себе чотири окремих завдання:

1. Створення пайплайнів для вилучення та очищення гетерогенних мобільних логів.
2. Програмування модулів векторизації та розрахунку синтетичних метрик.
3. Тренування моделі машинного навчання та крос-валідація.
4. Розрахунок значень Шеплі для інтерпретації та оптимізації порогу.

Завдання 1 та 3 за ступенем новизни відносяться до групи А — розробка нових алгоритмів/науково-дослідних моделей, завдання 2 та 4 — до групи Б — розробка типових/оригінальних рішень. За складністю алгоритми, які використовуються в завданнях 3 та 4, належать до групи 1 — моделювання та оптимізація, а в завданнях 1 та 2 — до групи 2 — облік, пошук, статистика. Для реалізації завдань 1 та 2 використовується масив змінної інформації, а завдання 3 та 4 використовують нормативно-довідкову інформацію.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як:

$$T_0 = T_p \cdot K_{\Pi} \cdot K_{\text{СК}} \cdot K_{\text{М}} \cdot K_{\text{СТ}} \cdot K_{\text{СТ.М}}, \quad (4.8)$$

де T_p — трудомісткість розробки ПП;

K_{Π} — поправочний коефіцієнт;

$K_{\text{СК}}$ — коефіцієнт на складність вхідної інформації;

$K_{\text{М}}$ — коефіцієнт рівня мови програмування;

$K_{\text{СТ}}$ — коефіцієнт використання стандартних модулів і прикладних програм;

$K_{\text{СТ.М}}$ — коефіцієнт стандартного математичного забезпечення.

Для першого завдання для обох варіантів, виходячи із норм часу для завдань розрахункового характеру ступеня новизни А та групи складності алгоритму 2, нормативна трудомісткість дорівнює: $T_p = 36$ людино-днів. Поправочний коефіцієнт на вид змінної інформації: $K_{\Pi} = 2.52$. Оскільки обробляються сирі лог-файли, складність контролю інформації висока: $K_{\text{ск}} = 1.08$. Програмування ведеться високорівневою мовою Python, тому $K_M = 1$. Використовуються нестандартні архітектурні рішення, тому $K_{\text{ст}} = 1$, $K_{\text{ст.м}} = 1$. Тоді загальна трудомісткість:

$$T_1 = 36 \cdot 2.52 \cdot 1.08 \cdot 1 \cdot 1 \cdot 1 = 97.98 \text{ людино-днів}$$

Для другого завдання для обох варіантів, алгоритм 2-ї групи складності, ступінь новизни Б: $T_p = 19$ людино-днів, $K_{\Pi} = 1.8$, $K_{\text{ск}} = 1.08$, $K_{\text{ст}} = 0.8$:

$$T_2 = 19 \cdot 1.8 \cdot 1.08 \cdot 1 \cdot 0.8 \cdot 1 = 29.55 \text{ людино-днів}$$

Етап для третього завдання відрізняється для варіантів.

Для Варіанта 1: алгоритм 1-ї групи складності, новизна А: $T_p = 90$ людино-днів. $K_{\Pi} = 1.7$, $K_{\text{ск}} = 1$. Оскільки LightGBM потребує складнішої попередньої обробки категоріальних ознак та ONE, $K_{\text{ст}} = 0.9$.

$$T_{3(1)} = 90 \cdot 1.7 \cdot 1 \cdot 1 \cdot 0.9 \cdot 1 = 137.7 \text{ людино-днів}$$

Для Варіанта 2: алгоритм 1-ї групи складності, новизна А: $T_p = 90$ людино-днів. $K_{\Pi} = 1.7$, $K_{\text{ск}} = 1$. CatBoost «з коробки» нативно обробляє гетерогенні дані, що дозволяє ширше використати стандартні пайплайни: $K_{\text{ст}} = 0.7$.

$$T_{3(2)} = 90 \cdot 1.7 \cdot 1 \cdot 1 \cdot 0.7 \cdot 1 = 107.1 \text{ людино-днів}$$

Для четвертого завдання для обох варіантів: алгоритм 1-ї групи складності, новизна Б: $T_p = 64$ людино-днів. $K_{\Pi} = 1.02$, $K_{\text{ск}} = 1$, $K_{\text{ст}} = 0.8$, оскільки використовується бібліотека shap:

$$T_4 = 65 \cdot 1.02 \cdot 1 \cdot 1 \cdot 0.8 \cdot 1 = 52.22 \text{ людино-днів}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації, переводимо в години, множачи на 8:

$$T_1 = (97.98 + 29.55 + 137.7 + 52.22) \cdot 8 = 2539.6 \text{ людино-годин}$$

$$T_2 = (97.98 + 29.55 + 107.1 + 52.22) \cdot 8 = 2294.8 \text{ людино-годин}$$

Найбільш високу трудомісткість має Варіант 1, через необхідність додаткового програмування для LightGBM.

В розробці беруть участь два спеціалісти: Machine Learning Engineer з окладом 90 000 грн. та Data Scientist з окладом 80 000 грн. Визначимо середню зарплату за годину за формулою:

$$C_{\text{ч}} = \frac{M}{T_m \cdot t} \text{ грн.}, \quad (4.9)$$

де M — місячний оклад працівників;

T_m — кількість робочих днів;

t — кількість робочих годин в день.

$$C_{\text{ч}} = \frac{90000+80000}{2 \cdot 21 \cdot 8} = 1\,349.21 \text{ грн.},$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{\text{зп}} = C_{\text{ч}} \cdot T_i \cdot K_{\text{д}}, \quad (4.10)$$

де $C_{\text{ч}}$ — величина погодинної оплати праці програміста;

T_i — трудомісткість відповідного завдання;

$K_{\text{д}}$ — норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить.

$$1. C_{\text{зп}} = 1\,349.21 \cdot 2539.6 \cdot 1.2 = 4\,111\,744.46 \text{ грн.}$$

$$2. C_{\text{зп}} = 1\,349.21 \cdot 2294.8 \cdot 1.2 = 3\,715\,400.53 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22% [32].

$$1. C_{\text{вд}} = C_{\text{зп}} \cdot 0.22 = 4\,111\,744.46 \cdot 0.22 = 904\,583.78 \text{ грн.}$$

$$2. C_{\text{вд}} = C_{\text{зп}} \cdot 0.22 = 3\,715\,400.53 \cdot 0.22 = 817\,388.12 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. ($C_{\text{м}}$)

Так як одна ЕОМ обслуговує ML-інженера з окладом 90000 грн., з коефіцієнтом зайнятості 0.2 то для однієї машини отримаємо[33]:

$$C_{\text{г}} = 12 \cdot M \cdot K_3 = 12 \cdot 90000 \cdot 0.2 = 216\,000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{\text{зп}} = C_{\text{г}} \cdot (1 + K_3) = 216\,000 \cdot 1.2 = 259\,200 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 259\,200 \cdot 0.22 = 57\,024 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості графічної станції 120 000 грн., $K_{\text{ТМ}} = 1.4$:

$$C_{\text{А}} = K_{\text{ТМ}} \cdot K_{\text{А}} \cdot C_{\text{ПР}} = 1.4 \cdot 0.25 \cdot 120\,000 = 42\,000 \text{ грн.,}$$

де $K_{\text{ТМ}}$ — коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$K_{\text{А}}$ — річна норма амортизації;

$C_{\text{ПР}}$ — договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_{\text{Р}} = K_{\text{ТМ}} \cdot C_{\text{ПР}} \cdot K_{\text{Р}} = 1.4 \cdot 120\,000 \cdot 0.08 = 13\,440 \text{ грн.,}$$

де $K_{\text{Р}}$ — відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{\text{ЕФ}} = (D_{\text{К}} - D_{\text{В}} - D_{\text{С}} - D_{\text{Р}}) \cdot t \cdot K_{\text{В}}. \quad (4.11)$$

$$T_{\text{ЕФ}} = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.8 = 1491.2 \text{ години,}$$

де $D_{\text{К}}$ — календарна кількість днів у році;

$D_{\text{В}}, D_{\text{С}}$ — відповідно кількість вихідних та святкових днів;

$D_{\text{Р}}$ — кількість днів планових ремонтів устаткування;

t — кількість робочих годин в день;

$K_{\text{В}}$ — коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} \cdot N_{\text{С}} \cdot K_{\text{З}} \cdot C_{\text{ЕН}} = 1491.2 \cdot 0.2 \cdot 0.5 \cdot 13.64 = 2\,034 \text{ грн.,}$$

де $N_{\text{С}}$ — середньо-споживча потужність приладу;

$K_{\text{З}}$ — коефіцієнтом зайнятості приладу;

$C_{\text{ЕН}}$ — тариф за 1 КВт-годин електроенергії, який складає 13.64 грн. [34].

Накладні витрати розраховуємо за формулою:

$$C_{\text{Н}} = C_{\text{ПР}} \cdot 0.67 = 120\,000 \cdot 0.67 = 80\,400 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{А}} + C_{\text{Р}} + C_{\text{ЕЛ}} + C_{\text{Н}}, \quad (4.12)$$

$$C_{\text{ЕКС}} = 259200 + 57024 + 42000 + 13440 + 2034 + 80400 = 454098 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = C_{\text{ЕКС}} / T_{\text{ЕФ}} = 454098 / 1491.2 = 304.52 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_{\text{М}} = C_{\text{М-Г}} \cdot T, \quad (4.13)$$

$$1. C_{\text{М}} = 304.52 \cdot 2539.6 = 773\,358.99 \text{ грн.}$$

$$2. C_{\text{М}} = 304.52 \cdot 2294.8 = 698\,812.50 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$1. C_{\text{Н}} = 773\,358.99 \cdot 0.67 = 518\,150.52 \text{ грн.}$$

$$2. C_{\text{Н}} = 698\,812.50 \cdot 0.67 = 468\,204.37 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{\text{ПП}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{М}} + C_{\text{Н}}, \quad (4.14)$$

$$1. C_{\text{ПП}} = 4\,111\,744.46 + 904\,583.78 + 773\,358.99 + 518\,150.52 = \\ = 6\,307\,837.75 \text{ грн.}$$

$$2. C_{\text{ПП}} = 3\,715\,400.53 + 817\,388.12 + 698\,812.50 + 468\,204.37 = \\ = 5\,699\,805.52 \text{ грн.}$$

4.7 Вибір кращого варіанту розробки за критерієм техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}} = \frac{K_{\text{К}}}{C_{\text{ПП}}}. \quad (4.15)$$

$$1. K_{\text{ТЕР}} = \frac{6.637}{6} 307\,837.75 = 1.0522 \cdot 10^{-6}.$$

$$2. K_{\text{ТЕР}} = \frac{8.102}{5} 699\,805.52 = 1.4215 \cdot 10^{-6}.$$

Як бачимо з отриманих результатів, найбільш ефективним є другий варіант реалізації програмного конвеєра з коефіцієнтом техніко-економічного рівня $K_{\text{ТЕР}} = 1.4215 \cdot 10^{-6}$. Цей показник виявився вищим, ніж у конкурентного алгоритму, що свідчить про його більшу доцільність для впровадження.

Після виконання функціонально-вартісного аналізу розроблюваного мікросервісу предиктивної аналітики, можна зробити висновок, що з альтернатив, які залишились після попереднього відбору, технічно та економічно оптимальним є другий варіант реалізації.

Перемога цього варіанта є цілком закономірною математично та економічно: він не лише продемонстрував вищий загальний показник технічної якості, але й виявився значно дешевшим у розробці. Зниження вартості створення системи досягається завдяки здатності обраного алгоритму нативно обробляти категоріальні та гетерогенні дані «з коробки». Це звільняє команду Data Science інженерів від необхідності програмувати складні модулі для додаткового енкодингу ознак, що суттєво скорочує найбільш дорогую статтю витрат — трудомісткість розробки.

Цей переможний варіант реалізації програмного продукту має такі архітектурні параметри:

- 1) за обробку даних відповідає фреймворк Polars (використання багатопотокової колонкової архітектури для запобігання переповненню RAM);
- 2) модель машинного навчання CatBoost;
- 3) інтерпретація результатів за допомогою SHAP-аналізу;
- 4) інтеграція з САРІ з динамічним калібруванням порогу ймовірності.

Запропонований варіант виконання програмного комплексу дає бізнесу безкомпромісну швидкість інференсу, що повністю унеможливорює накопичення черг та таймауту при передачі сигналів у зовнішні рекламні мережі. Крім того, система гарантує високу точність класифікації в умовах дисбалансу класів і забезпечує повну прозорість логіки прийняття рішень для

стейкхолдерів, тим самим мінімізуючи управлінські та фінансові ризики компанії.

4.8 Висновки до розділу 4

На основі проведеного функціонально-вартісного аналізу вартості розробки мікросервісу предиктивної аналітики можна зробити обґрунтований висновок про високий потенціал даного проєкту як для інтеграції у діючі маркетингові екосистеми, так і для автономної комерціалізації у форматі SaaS-рішення.

Застосування методології ФВА дозволило комплексно оцінити економічні перспективи проєкту, забезпечивши суворий баланс між оптимізацією бюджету та максимально ефективним використанням апаратних і людських ресурсів. У процесі аналізу було сформовано та досліджено кілька альтернативних сценаріїв виконання основних функцій продукту. Це дозволило адаптувати фінальне архітектурне рішення під жорсткі технічні вимоги високонавантажених систем та наявні економічні обмеження.

У результаті обчислень доведено, що обраний програмний продукт не тільки повністю задовольняє критичні бізнес-вимоги щодо точності прогнозування та швидкості інференсу, але й є найбільш економічно вигідним. Мінімізація початкових витрат на розробку була досягнута завдяки здатності алгоритму нативно обробляти гетерогенні дані, що радикально скоротило трудомісткість етапу Feature Engineering для команди Data Science.

Враховуючи високу вартість залучення висококваліфікованих інженерних кадрів та оренди GPU-серверів, надзвичайно важливим здобутком є те, що оптимізація обраних технологій дозволила знизити загальну вартість розробки без жодних компромісів у якості алгоритму. Висока бізнес-інтерпретованість моделі та мінімальна операційна затримка під час роботи з

SAPİ гарантують швидке повернення інвестицій за рахунок превентивного блокування неефективних рекламних витрат. Подальший пошук шляхів для зниження операційних витрат та підвищення ефективності проекту може бути спрямований на оптимізацію хмарної інфраструктури та автоматизацію процесів донавчання розробленої моделі.

ВИСНОВКИ

У процесі виконання дипломної роботи було вирішено актуальну науково-прикладну задачу розробки системи предиктивної аналітики для прогнозування ймовірності конверсії користувачів мобільного застосунку. Робота охопила повний цикл проєктування Data Science рішень: від побудови масштабованого конвеєра обробки гетерогенних логів та конструювання складних поведінкових метрик до тренування ансамблів градієнтного бустингу та їх функціонально-вартісного обґрунтування.

Комплексний підхід до інженерії даних дозволив розв'язати проблему екстремального споживання обчислювальних ресурсів за рахунок відмови від традиційних рядкових архітектур на користь фреймворку лінійних обчислень Polars та колонкового формату Apache Arrow. Окремим науковим здобутком роботи є математична формалізація та алгоритмізація когнітивного навантаження користувачів. Було успішно розроблено індекси фрустрації, а також адаптовано методи обробки природної мови для виявлення прихованих секвенціальних патернів у розріджених хронологічних даних. Застосування методів просторової нормалізації та дисперсійної фільтрації дозволило нівелювати «прокляття розмірності» при формуванні фінального датасету.

На основі згенерованого високорозмірного простору ознак було проведено порівняльний експериментальний аналіз сучасних алгоритмів градієнтного бустингу XGBoost, LightGBM та CatBoost із застосуванням методології хронологічної крос-валідації. Отримані результати підтверджують високу ефективність GBDT-моделей у задачах бінарної класифікації з екстремальним дисбалансом класів. Хоча LightGBM продемонстрував найвищі показники за метриками F1-Score та MCC, оптимальним вибором для промислової мікросервісної експлуатації було визнано архітектуру CatBoost. Вона забезпечила найшвидший час інференсу, ефективну нативну обробку висококардинальних категоріальних ознак через

алгоритм Ordered Target Statistics та високу стійкість до перенавчання завдяки топології симетричних дерев.

Завдяки імплементації методології SHAP вдалося подолати проблему алгоритмічної «чорної скриньки» та отримати як глобальну, так і локальну інтерпретацію предикцій. Аналіз каскадних діаграм та графіків бджолиного рою об'єктивно довів, що надмірне когнітивне навантаження, системні бар'єри та агресивна push-комунікація діють як першопричини відмови від монетизації. Для інтеграції моделі з рекламними мережами через серверний протокол Conversions API було математично обґрунтовано необхідність зміщення порогу прийняття рішень. Такий підхід гарантує максимізацію чистоти сигналу, формуючи високоочищений потік "синтетичних конверсій", що унеможливорює деградацію алгоритмів таргетингу рекламних платформ.

Проведений функціонально-вартісний аналіз повністю підтвердив економічну доцільність розробленого програмного рішення. Обраний варіант архітектури дозволив не лише мінімізувати транзитне споживання оперативної пам'яті, але й суттєво скоротити трудомісткість розробки за рахунок нативної обробки гетерогенних даних, що забезпечило найвищий коефіцієнт техніко-економічного рівня. Таким чином, результати дослідження мають як теоретичну цінність у площині математичного моделювання поведінки, так і безпосереднє практичне значення для автоматизації performance-маркетингу та підвищення рентабельності мобільних ІТ-продуктів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. State of Mobile 2026: The Industry-Defining Report. SensorTower. 2026.
URL: <https://sensortower.com/report/state-of-mobile-2026> (Last accessed: 02.06.2026).
2. Eyal N., Hoover N. Hooked: How to build habit-forming products. Portfolio, 2014. 256 p. URL: <https://www.nirandfar.com/hooked/> (Last accessed: 02.06.2026).
3. Yuan H., Li S., Rusconi P. *Cognitive effort and frustration in user interface navigation*. *ACM Transactions on Computer-Human Interaction*. 2021. Vol. 28, No. 3. P. 114–128. URL: <https://dl.acm.org/doi/10.1145/3447534> (Last accessed: 02.06.2026).
4. Недашківська Н. І. Методи і моделі інтелектуального аналізу даних: практикум: навч. посіб. для студ. спеціальності 122 «Комп'ютерні науки», освітньої програми «Системи і методи штучного інтелекту» / КПІ ім. Ігоря Сікорського. Київ: КПІ ім. Ігоря Сікорського, 2019. 70 с.
URL: <https://ela.kpi.ua/handle/123456789/53764> (дата звернення: 02.06.2026).
5. Flach P. A., Kull M. *Precision-Recall-Gain Curves: PR analysis done right*. *Advances in Neural Information Processing Systems*. 2015. Vol. 28. P. 838–846. URL: <https://proceedings.neurips.cc/paper/2015/hash/33e8075e9970de0cfea955afd4644bb2-Abstract.html> (Last accessed: 02.06.2026).
6. Chawla N. V. et. al. *SMOTE: synthetic minority over-sampling technique*. *Journal of Artificial Intelligence Research*. 2002. Vol. 16. P. 321–357. DOI: [10.1613/jair.953](https://doi.org/10.1613/jair.953).
7. Chicco D., Jurman G. *The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation*. *BMC Genomics*. 2020. Vol. 21, No. 6. URL:

- <https://link.springer.com/article/10.1186/s12864-019-6413-7> (Last accessed: 02.06.2026).
8. McKinney W. Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference*. Tacoma, Washington, USA. 2010. Vol. 445. P. 51–56. DOI: [10.25080/Majora-92bf1922-00a](https://doi.org/10.25080/Majora-92bf1922-00a).
 9. Polars: Lightning-fast DataFrame library for Rust and Python. 2026. URL: <https://docs.pola.rs/> (Last accessed: 02.06.2026).
 10. Apache Arrow Format Specification. Apache Software Foundation. 2025. URL: <https://arrow.apache.org/docs/format/#specifications> (Last accessed: 02.06.2026).
 11. Manning C. D., Raghavan P., Schütze H. Introduction to information retrieval. Cambridge University Press, 2009. 581 p. URL: <https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf> (Last accessed: 02.06.2026).
 12. Friedman J. H. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*. 2001. P. 1189–1232. DOI: [10.1214/aos/1013203451](https://doi.org/10.1214/aos/1013203451).
 13. Chen T., Guestrin C. XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, California, USA. 2016. P. 785–794. DOI: <https://doi.org/10.1145/2939672.2939785> (Last accessed: 02.06.2026).
 14. Prokhorenkova L. et al. CatBoost: unbiased boosting with categorical features. *Advances in Neural Information Processing Systems*. 2018. Vol. 31. P. 6638–6648. DOI: <https://arxiv.org/abs/1706.09516> (Last accessed: 02.06.2026).
 15. Ke G. et. al. LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*. 2017. Vol. 30. P. 3146–3154. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf (Last accessed: 02.06.2026).

16. Meta Conversions API (CAPI) Developer Documentation. Meta Platforms Inc. 2026. URL: <https://developers.facebook.com/documentation/ads-commerce/conversions-api> (Last accessed: 02.06.2026).
17. Amdahl G. M. Validity of the single processor approach to achieving large scale computing capabilities. AFIPS Spring Joint Computer Conference.  New York, NY, USA. 1967. P. 483–485. DOI: [10.1145/1465482.1465560](https://doi.org/10.1145/1465482.1465560).
18. Blumofe R. D., Leiserson C. E. Scheduling multithreaded computations by work stealing. *Journal of the ACM*. 1999. Vol. 46, No. 5. P. 720–748. DOI: [10.1145/324133.324234](https://doi.org/10.1145/324133.324234).
19. Iglewicz B., Hoaglin D. C. How to detect and handle outliers. ASQC Quality Press, 1993. Vol. 16, 99 p. URL: https://books.google.com.ua/books/about/Volume_16_How_to_Detect_and_Handle_Outli.html?id=FuuiEAAAQBAJ&redir_esc=y (Last accessed: 02.06.2026).
20. Shannon C. E. A mathematical theory of communication. *Bell System Technical Journal*. 1948. Vol. 27. P. 379–423. URL: <https://ieeexplore.ieee.org/document/6773024> (Last accessed: 02.06.2026).
21. Hastie T., Tibshirani R., Friedman J. The elements of statistical learning: Data mining, inference, and prediction. Springer, 2009. 745 p. URL: <https://hastie.su.domains/ElemStatLearn/> (Last accessed: 02.06.2026).
22. Arlot S., Celisse A. A survey of cross-validation procedures for model selection. *Statistics Surveys*. 2009. Vol. 4. P. 40–79. DOI: [10.1214/09-SS054](https://doi.org/10.1214/09-SS054).
23. Sulandari W. et. al. Implementing Time Series Cross Validation to Evaluate the Forecasting Model Performance. *KnE Life Sciences*. 2024. P. 229–238. DOI: [10.18502/cls.v8i1.15584](https://doi.org/10.18502/cls.v8i1.15584) (Last accessed: 02.06.2026).
24. Lundberg S., Lee S.-I. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*. 2017. Vol. 30. P. 4765–4774. DOI: <https://doi.org/10.48550/arXiv.1705.07874> (Last accessed: 02.06.2026).

25. User Privacy and Data Use. Apple Developer. URL: <https://developer.apple.com/app-store/user-privacy-and-data-use/> (Last accessed: 02.06.2026).
26. Platt J. C. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*. 1999. Vol. 10, No. 3. P. 61–74. DOI: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.1639> (Last accessed: 02.06.2026).
27. Zadrozny B., Elkan C. Transforming classifier scores into accurate multiclass probability estimates. *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Edmonton, Alberta, Canada. July 23–26, 2002. P. 694–699. DOI: <https://doi.org/10.1145/775047.775151> (Last accessed: 02.06.2026).
28. Data Version Control (DVC) Documentation. URL: <https://dvc.org/doc> (Last accessed: 02.06.2026).
29. Virtanen P. et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*. 2019. Vol. 17, No. 3. P. 261–272. DOI: [10.1038/s41592-019-0686-2](https://doi.org/10.1038/s41592-019-0686-2).
30. Pedregosa F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*. 2011. Vol. 12. P. 2825–2830. URL: <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html> (Last accessed: 02.06.2026).
31. Економіка та організація виробництва. Методичні вказівки до виконання економічного розділу дипломних робіт / уклад. Н. В. Рощина. Київ: КПІ ім. Ігоря Сікорського, 2021. 45 с.
32. Довідник показників. Єдиний соціальний внесок. Дебет-Кредит. URL: <https://services.dtkr.ua/catalogues/indexes/13> (дата звернення: 02.06.2026).
33. Мінімальна заробітна плата в Україні. Мінфін. URL: <https://index.minfin.com.ua/labour/salary/min/> (дата звернення: 02.06.2026).

34. Тарифи на електроенергію для бізнесу. YASNO. URL: <https://yasno.com.ua/business/b2b-tariffs> (дата звернення: 02.06.2026).
35. Зайченко Ю.П. Теорія прийняття рішень. Київ: Видавництво «КПІ», 2014. 412 с.

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО КОДУ

```

# CONSOLIDATED PIPELINE: User Conversion Prediction for Mobile App
from __future__ import annotations

import gc
import glob
import json
import math
import os
from datetime import date, datetime, timedelta
from pathlib import Path
import shutil
import sys
import time
import traceback
import tracemalloc
import warnings
from urllib.parse import quote_plus

os.environ["POLARS_MAX_THREADS"] = "4"
os.environ["OMP_NUM_THREADS"] = "1"
os.environ["MKL_NUM_THREADS"] = "1"

from catboost import CatBoostClassifier, Pool
from dateutil.relativedelta import relativedelta
from dotenv import load_dotenv
from google.cloud import bigquery
from google.oauth2 import service_account
import joblib
import lightgbm as lgb
import matplotlib.pyplot as plt
import mysql.connector
import numpy as np
import pandas as pd
import polars as pl
import psutil
import pyarrow as pa
import pyarrow.parquet as pq
import rootutils
import seaborn as sns
import shap
from sklearn.calibration import calibration_curve
from sklearn.metrics import (
    accuracy_score, average_precision_score, brier_score_loss,
    classification_report, confusion_matrix, f1_score, log_loss,
    matthews_corrcoef, mean_squared_error, precision_recall_curve,
    precision_score, r2_score, recall_score, roc_auc_score,
)
import xgboost as xgb

rootutils.setup_root(Path.cwd(), indicator=".project-root", pythonpath=True)
load_dotenv()

# CONFIG
ROOT = Path(rootutils.setup_root(Path.cwd(), indicator=".project-root",
pythonpath=True))
DATA_DIR = ROOT / "data"
RAW_DATA_DIR = DATA_DIR / "raw"

```

```

PROCESSED_DATA_DIR = DATA_DIR / "processed"
ARTIFACTS_DIR = DATA_DIR / "artifacts"
MODELS_DIR = ROOT / "models"
REPORTS_DIR = ROOT / "reports"
SRC_DIR = ROOT / "src"
SQL_DIR = SRC_DIR / "sql_queries"

for _d in [DATA_DIR, RAW_DATA_DIR, PROCESSED_DATA_DIR, ARTIFACTS_DIR,
MODELS_DIR, REPORTS_DIR]:
    _d.mkdir(parents=True, exist_ok=True)

cities_csv_path = DATA_DIR / "largest-cities-by-population-2025.csv"
bq_credentials_path = ROOT / "bq_plantin_app_cred.json"
unified_stats_path = DATA_DIR / "unified_stats_history_6h.parquet"
static_artifacts_path = ARTIFACTS_DIR / "static_features.json"
tfidf_vocab_path = ARTIFACTS_DIR / "tfidf_vocab.json"
static_cohort_stats_path = ARTIFACTS_DIR /
"static_cohort_stats_baseline.parquet"

MYSQL_HOST = os.environ.get("host_plantin")
MYSQL_DB = os.environ.get("database_plantin")
MYSQL_USER = os.environ.get("user")
MYSQL_PASSWORD = os.environ.get("password_plantin")
BQ_PROJECT_ID = "plantin-app"

user_col = "user_id"
target_col = "is Converted"
date_col = "original_purchase_date"
install_date_col = "install_date"
event_time_col = "event_time"
event_col = "event_type"
cohort_cols = ["original_purchase_date", "country", "media_source"]

cat_cols = ["country", "device_type", "source", "cmp_type", "creo_type"]
num_cols = ["is_big_city", "ctr", "c2i", "cpm", "cpc", "cpi", "spend_usd",
            "price", "trial_price", "trial_period", "period_days",
            "app_version"]

trial_window_hours = 6
pre_trial_recent_hours = 24
session_break_seconds = 1800

interaction_pairs = [
    ("country", "device_type"), ("source", "country"),
    ("device_type", "source"), ("device_type", "is_big_city"),
]
top_n_config = {"country": 10, "device_type": 5, "source": 5}

top_k = 50

conversion_pairs = [
    ("app_install", "plant_open"), ("app_install", "disease_open"),
    ("app_install", "plant_added"), ("app_install", "onboarding_finished"),
    ("app_install", "authorized"), ("app_install", "ss_open"),
    ("visit_identification", "plant_open"), ("visit_identification",
"plant_added"),
    ("visit_diagnosis", "disease_open"), ("plant_open", "plant_added"),
    ("plant_added", "care_performed"), ("sent_push", "app_launch"),
    ("sent_push_care_reminder", "app_launch"), ("visit_settings",
"disable_push"),
    ("botanist_show_form", "botanist_init"), ("botanist_init",
"botanist_asked"),
]

```

```

target_events = [
    "session_start", "app_finished_loading", "app_launch",
    "pop_up_about_updates_open",
    "season_pass_visit", "visit_my_plant", "user_plant_extra_setting_save",
    "user_plant_extra_setting_configure", "watering_calculator_show",
    "space_created",
    "watering_calculator_save", "user_plant_care_performed",
    "plant_wishlist_add",
    "space_open", "light_meter_open", "collected_plant", "health_check_save",
    "user_plant_remove", "visit_improve_care_plan", "onboarding_finished",
    "light_meter_save_result", "detailed_review_agreed", "image_open",
    "user_plant_record_rain_change", "visit_my_plant_tab", "visit_search",
    "plant_requested", "watering_calculator_deny", "video_play",
    "visit_user_plant_care_settings", "care_tips_video_play",
    "care_tips_video_close",
    "plant_info_opened", "plant_wiki_open", "ff_authorized",
    "plant_open_tab_general_info",
    "startup_user_screen_queue", "visit_ar", "ff_onboarding_loaded",
    "push_notifications_subscribed", "diseases_list",
    "ff_onboarding_finished",
    "plant_added", "camera_loaded", "search_failed", "onboarding_step",
    "diseases_questionnaire_skip", "ask_botanist_delete_request",
    "diseases_identification_failed", "visit_treatment_tab",
    "ask_botanist_show_form",
    "visit_feed", "moon_calendar_open", "onboarding_skip",
    "visit_profile_settings",
    "app_closed_url", "onboarding_started", "rate_app_success",
    "gallery_open",
    "push_notification_confirmation", "plant_open_main_tabs",
    "visit_my_plants",
    "app_open_url", "gallery_image_view", "plant_open_tab",
    "visit_disease_tab",
    "feedback_success", "app_error", "push_notifications_popup_show",
    "diseases_treatment_open", "diseases_save", "push_notification_open_app",
    "main_identification_tap", "push_notification_request_access",
    "visit_diseases",
    "visit_diagnosis", "diseases_plant_added_success", "tutorial_finish",
    "push_notification_denied", "visit_settings", "tutorial_start",
    "visit_menu",
    "diseases_questionnaire_finished", "plant_open",
    "diseases_questionnaire",
    "search_success", "diseases_search_plant", "visit_identification",
    "diseases_open",
    "tutorial_skip", "tutorial_visit_identify",
    "permission_request_completed",
    "ff_complete", "diseases_identification_success", "visit_snap_tips",
]

forbidden_event_cats = ["purchase", "subscription"]

event_cats = {
    "auth": {
        "positive": ["auth_login", "auth_reset_password_success",
    "auth_signup"],
        "negative": ["auth_logout"],
        "neutral": ["auth_reset_password_request",
    "auth_request_reset_password",
        "auth_reset_password_check_email",
    "auth_reset_password_check_email_tap",
        "auth_reset_password_initiated", "auth_sign_in_tap",
    "auth_sign_in_view",
        "auth_sign_up_tap", "auth_sign_up_view"],
    },

```

```

    "blog": {"positive": ["visit_blog", "blog_visit_article"], "negative":
[], "neutral": []},
    "botanist": {
        "positive": ["ask_botanist_show_form", "ask_botanist_initiated",
"ask_botanist_asked",
            "ask_botanist_show_answer", "ask_botanist_answers",
"ask_botanist_answer_commented",
            "ask_botanist_choose_type_initiated",
"ask_botanist_choose_type_finished"],
        "negative": ["ask_botanist_cancelled", "ask_botanist_failed"],
        "neutral": [],
    },
    "disease": {
        "positive": ["visit_diagnosis", "diseases_identification_initiated",
            "diseases_identification_success", "diseases_open"],
        "negative": ["diseases_identification_failed"],
        "neutral": ["diseases_list"],
    },
    "feed": {
        "positive": ["visit_feed", "feed_item_create", "feed_item_dislike",
"feed_item_like",
            "feed_item_open", "feed_item_update",
"feed_open_profile", "feed_open_tag",
            "feed_open_user", "social_comment_create",
"social_comment_dislike",
            "social_comment_like", "social_comment_remove",
"social_comment_update",
            "feed_open_plant", "feed_search_initiated",
"feed_search_success",
            "user_update_avatar", "user_update_email",
"user_update_fullname", "user_update_username"],
        "negative": ["feed_search_failed", "user_remove_avatar",
"feed_item_remove", "feed_item_report"],
        "neutral": ["visit_profile_settings", "social_comment_dismark_best",
"social_comment_mark_best"],
    },
    "ff": {
        "positive": ["ff_authorized", "ff_complete"],
        "negative": ["ff_auth_skip"],
        "neutral": ["ff_onboarding_continue_tap", "ff_onboarding_finished",
"ff_onboarding_loaded", "ff_onboarding_screen_view"],
    },
    "lightmeter": {"positive": ["apps_open", "light_meter_open",
"light_meter_save_result"], "negative": [], "neutral": []},
    "menu": {
        "positive": ["expert_guide_download", "themes_purchase_initiated",
"themes_purchased"],
        "negative": [],
        "neutral": ["visit_settings", "visit_theme_store", "theme_viewed",
"theme_updated",
            "theme_proposed", "visit_menu",
"dashboard_achievements_explain_open", "dashboard_event_tap"],
    },
    "moon": {"positive": ["moon_calendar_open", "moon_calendar_change_date"],
"negative": [], "neutral": []},
    "mushroom": {
        "positive": ["mushroom_open",
"mushrooms_upload_offline_identifications_initiated"],
        "negative": [],
        "neutral": ["offline_photo_delete"],
    },
    "offline": {
        "positive": ["mushrooms_upload_offline_identifications_initiated",
"offline_photo_added",

```

```

        "offline_search_initiated", "offline_search_success",
"offline_pop_up_open"],
        "negative": ["offline_pop_up_close"],
        "neutral": ["offline_photo_delete", "offline_search_failed"],
    },
    "onboarding": {
        "positive": ["ff_onboarding_loaded", "ff_onboarding_screen_view",
"onboarding_started",
            "onboarding_step", "onboarding_finished"],
        "negative": ["onboarding_skip"],
        "neutral": ["ff_onboarding_continue_tap", "ff_onboarding_finished",
"onboarding_purchase"],
    },
    "plant": {
        "positive": ["plant_wishlist_open", "plant_wishlist_add",
"plant_added", "plant_open",
            "image_open", "plant_page_faq_tap"],
        "negative": ["plant_requested"],
        "neutral": ["plant_wishlist_remove", "offline_photo_delete"],
    },
    "pop_up": {
        "positive": ["offline_pop_up_open", "pop_up_about_updates_open",
"survey_pop_up_answer"],
        "negative": ["offline_pop_up_close", "pop_up_about_updates_close",
"survey_pop_up_close"],
        "neutral": ["push_notification_request_access",
"survey_pop_up_show"],
    },
    "purchase": {
        "positive": ["onboarding_purchase",
"subscription_upgrade_to_lifetime_initiated",
            "purchase_paid", "purchase_restored",
"purchase_transactions_finished", "purchase_verified"],
        "negative": ["purchase_failed", "purchase_cancelled"],
        "neutral": ["_purchase_finished_internal"],
    },
    "push": {
        "positive": ["push_notification_open_app",
"push_notification_received", "push_notifications_subscribed"],
        "negative": ["push_notification_change_preference",
"push_notification_denied"],
        "neutral": ["push_notification_request_access", "sent_push"],
    },
    "rate": {"positive": ["rate_app_success"], "negative": [], "neutral":
["rate_app_initiated"]},
    "review": {"positive": ["detailed_review_agreed"], "negative": [],
"neutral": ["detailed_review_requested"]},
    "search": {
        "positive": ["main_identification_tap", "visit_identification",
"search_initiated", "search_success",
            "search_rated", "visit_ar", "ar_session_restarted",
"visit_search", "visit_search_history",
            "collected_plant_push_open",
"visit_image_search_history", "plant_open",
            "offline_search_initiated", "offline_search_success"],
        "negative": ["search_failed", "offline_search_failed"],
        "neutral": ["offline_photo_delete"],
    },
    "season_pass": {
        "positive": ["season_pass_visit", "season_pass_purchase_initiated",
            "season_pass_purchased", "season_pass_apply_theme"],
        "negative": [], "neutral": [],
    },
    "settings": {

```

```

        "positive": ["visit_profile_settings", "user_update_bio", "faq_read",
                     "settings_change_notification_time",
"subscription_upgrade_to_lifetime_initiated"],
        "negative": ["subscription_cancel_initiated",
"subscription_cancel_persuasion",
                     "subscription_cancel_reasoned",
"subscription_cancel_details",
                     "subscription_cancel_end",
"subscription_cancel_see_offer",
                     "delete_account_initiated", "delete_account_success",
"delete_account",
                     "delete_account_successt",
"push_notification_change_preference"],
        "neutral": [],
    },
    "space": {
        "positive": ["space_add_plant_initiated", "search_open_gallery",
"space_created",
                     "space_open", "space_updated", "spaces_reordered",
"space_change_view"],
        "negative": ["space_removed"],
        "neutral": [],
    },
    "subscription": {
        "positive": ["subscription_purchase_initiated",
"subscription_purchase_finished",
                     "subscription_upgrade_to_lifetime_initiated"],
        "negative": ["subscription_cancel_initiated",
"subscription_cancel_persuasion",
                     "subscription_cancel_reasoned",
"subscription_cancel_details",
                     "subscription_cancel_end",
"subscription_cancel_see_offer",
                     "subscription_purchase_cancelled",
"subscription_purchase_failed",
                     "subscription_purchase_restored",
"subscription_propose_on_close"],
        "neutral": ["subscription_opened", "subscription_closed"],
    },
    "survey": {
        "positive": ["survey_pop_up_show", "survey_pop_up_answer"],
        "negative": ["survey_pop_up_close"],
        "neutral": [],
    },
    "system": {
        "positive": ["app_install", "app_launch", "app_finished_loading",
"app_user_registration",
                     "permission_request_initiated",
"permission_request_completed"],
        "negative": ["app_error"],
        "neutral": ["app_language_change", "app_location_request",
"app_reported_geolocation",
                     "app_open_quick_action", "app_terminate"],
    },
    "user_plant": {
        "positive": ["user_plant_care_performed",
"user_plant_extra_setting_configure",
                     "user_plant_extra_setting_save",
"user_plant_history_image_added",
                     "user_plant_history_note_added",
"user_plant_history_note_changed",
                     "user_plant_image_updated",
"user_plant_show_weather_warning",

```

```

        "user_plant_update", "user_plant_record_rain_change",
"visit_my_plant"],
        "negative": ["user_plant_remove_initiated", "user_plant_remove",
"user_plant_retire",
        "user_plant_image_removed",
"user_plant_care_delete_initiated",
        "user_plant_care_delete_success",
"user_plant_care_snoozed",
        "user_plant_history_removed_item"],
        "neutral": ["user_plant_return_from_retire",
"user_plant_care_changed_description"],
    },
    "w2a": {"positive": [], "negative": [], "neutral": ["w2a_attribution"]},
    "water_calculator": {
        "positive": ["apps_open", "watering_calculator_propose",
"watering_calculator_pot_meter_open",
        "watering_calculator_pot_meter_success",
"watering_calculator_show", "watering_calculator_save"],
        "negative": ["watering_calculator_deny"],
        "neutral": [],
    },
    "widget": {"positive": ["widget_open_app"], "negative": [], "neutral":
[]},
    "wishlist": {"positive": ["plant_wishlist_open", "plant_wishlist_add"],
"negative": [], "neutral": []},
}

EVENT_MAPPING_DATA = [
    {"event_type": ev, "category": cat, "sentiment": sent}
    for cat, sents in event_cats.items()
    for sent, evs in sents.items()
    for ev in evs
]
ALL_CATEGORIES = list(event_cats.keys())

# DATA LOADING
def _get_mysql_engine():
    from sqlalchemy import create_engine
    pw = quote_plus(MYSQL_PASSWORD)
    user = quote_plus(MYSQL_USER)
    return
    create_engine(f"mysql+mysqlconnector://{user}:{pw}@{MYSQL_HOST}/{MYSQL_DB}")

def _load_sql(filename: str) -> str:
    path = SQL_DIR / filename
    if not path.exists():
        raise FileNotFoundError(f"SQL file not found: {path}")
    return path.read_text()

def fetch_event_data(user_ids: list, start_date, end_date) -> pl.DataFrame:
    if not user_ids:
        return pl.DataFrame()
    credentials =
service_account.Credentials.from_service_account_file(bq_credentials_path)
    client = bigquery.Client(credentials=credentials, project=BQ_PROJECT_ID)
    ids_str = ",".join([f'"{uid}"' for uid in user_ids])
    query = f"""
        SELECT user_id, device_type, event_time, event_type, event_properties
        FROM `Amplitude.EVENTS_426968`
        WHERE user_id IN ({ids_str})
    """

```

```

        AND event_time >= '{start_date.strftime("%Y-%m-%d %H:%M:%S")}'
        AND event_time <= '{end_date.strftime("%Y-%m-%d %H:%M:%S")}'
        AND event_time <= CURRENT_TIMESTAMP()
    """
    try:
        job = client.query(query)
        df_arrow = job.to_arrow()
        print(f"    BQ: {(job.total_bytes_processed or 0) / 1e6:.2f} MB
processed.")
        return pl.from_arrow(df_arrow) if df_arrow.num_rows > 0 else
pl.DataFrame()
    except Exception as e:
        print(f"BQ error: {e}")
        return pl.DataFrame()

def _join_cities(lf: pl.LazyFrame) -> pl.LazyFrame:
    if cities_csv_path.exists():
        lf_cities = pl.scan_csv(cities_csv_path).select(["city"]).unique()
        return (lf.join(lf_cities, on="city", how="left")
            .with_columns(pl.when(pl.col("city_right").is_not_null()).the
n(1).otherwise(0).alias("is_big_city"))
            .drop(["city", "city_right"]))
        return lf.with_columns(pl.lit(0).alias("is_big_city"))

def fetch_full_user_data(start_date: datetime, end_date: datetime, mode: str
= "new_trials") -> pl.DataFrame:
    from sqlalchemy import text
    marketing_query = _load_sql("marketing_data.sql")
    is_converted_sql = ("if(us2.purchase_date_ms is not null or
us3.purchase_date_ms is not null, 1, 0)"
        if mode == "train" else "NULL")
    final_query = marketing_query.format(
        is_converted_logic=is_converted_sql,
        start_date=start_date.strftime("%Y-%m-%d %H:%M:%S"),
        end_date=end_date.strftime("%Y-%m-%d %H:%M:%S"),
    )
    try:
        engine = _get_mysql_engine()
        with engine.connect() as conn:
            df_pandas = pd.read_sql(text(final_query), conn)
    except Exception as e:
        print(f"MySQL error: {e}")
        return pl.DataFrame()
    if df_pandas.empty:
        return pl.DataFrame()

    df_pandas["app_version"] = pd.to_numeric(
        df_pandas["app_version"].astype(str).str.replace(r"[.]", "",
regex=True), errors="coerce"
    ).fillna(0).astype("int64")
    lf_marketing = _join_cities(pl.from_pandas(df_pandas).lazy())
    df_marketing = lf_marketing.collect()

    user_ids = df_pandas["user_id"].unique().tolist()
    min_date = df_marketing["install_time"].min() - timedelta(hours=1)
    max_date = df_marketing["install_time"].max() + timedelta(hours=48)
    df_events = fetch_event_data(user_ids, min_date, max_date)
    if df_events.height == 0:
        return pl.DataFrame()

    lf_final = (
        df_marketing.lazy()

```

```

        .join(df_events.lazy().with_columns([pl.col("user_id").cast(pl.Int64)
,
pl.col("event_time").cast(pl.Datetime("us"))]),
            on="user_id", how="inner")
        .with_columns(pl.coalesce(["original_purchase_date",
"install_time"]).alias("start_point"))
        .filter(pl.col("event_time") <= pl.col("start_point") +
pl.duration(hours=24))
        .filter(~pl.col("event_type").str.contains("subscription|purchase|cancel|install", literal=False))
        .drop("start_point")
    )
    return lf_final.collect()

def fetch_full_history() -> pl.LazyFrame:
    data_path = RAW_DATA_DIR / "full_historical_data.parquet"
    if not data_path.exists():
        raise FileNotFoundError(f"File not found: {data_path}. Run
HistoricalDataLoader first.")
    lf = pl.scan_parquet(data_path)
    cols = lf.collect_schema().names()
    if "source" in cols and "media_source" not in cols:
        lf = lf.rename({"source": "media_source"})
    return lf.with_columns([
        pl.col("user_id").cast(pl.Int64),
        pl.col("event_time").cast(pl.Datetime("us")),
        pl.col("original_purchase_date").cast(pl.Datetime("us")),
        pl.col("install_time").cast(pl.Datetime("us")),
        pl.col("media_source").fill_null("unknown"),
        pl.col("country").fill_null("unknown"),
        pl.col("is_converted").cast(pl.Int8).fill_null(0) if "is_converted"
in cols else pl.lit(0).alias("is_converted"),
    ])

# HISTORICAL ETL
class HistoricalDataLoader:
    def __init__(self):
        self.marketing_path = DATA_DIR / "full_marketing_export.parquet"
        self.merged_path = RAW_DATA_DIR / "full_historical_data.parquet"

        self.bq_client = None
        try:
            cred_path = str(bq_credentials_path)
            if os.path.exists(cred_path):
                creds =
service_account.Credentials.from_service_account_file(cred_path)
                self.bq_client = bigquery.Client(credentials=creds,
project=BQ_PROJECT_ID)
            else:
                self.bq_client = bigquery.Client(project=BQ_PROJECT_ID)
        except Exception as e:
            print(f"BQ init failed: {e}")

    def extract_product_data(self, start_date: datetime, final_end_date:
datetime, split_date: datetime):
        if not self.bq_client:
            return
        periods = []
        current = start_date
        while current < split_date:

```

```

        next_d = current + pd.DateOffset(months=1)
        if next_d > split_date:
            next_d = split_date
        periods.append({"start": current, "end": next_d,
                        "file": DATA_DIR /
f"full_product_export_{current.strftime('%Y_%m')}.parquet"})
        current = next_d
        while current < final_end_date:
            next_d = current + timedelta(weeks=1)
            periods.append({"start": current, "end": next_d,
                            "file": DATA_DIR /
f"full_product_export_{current.strftime('%Y_%m_%d')}.parquet"})
            current = next_d

        query_tpl = """
        SELECT user_id, device_type, event_time, event_type,
event_properties
        FROM `Amplitude.EVENTS_426968`
        WHERE event_time >= '{start}' AND event_time < '{end}'
        AND user_id IS NOT NULL
        AND REGEXP_CONTAINS(event_type,
r'(?i)subscription|purchase|cancel|install')
        """
        for p in periods:
            if p["file"].exists():
                continue
            query = query_tpl.format(start=p["start"].strftime("%Y-%m-%d"),
end=p["end"].strftime("%Y-%m-%d"))
            try:
                job = self.bq_client.query(query)
                writer = None
                rows = 0
                for batch in
job.result().to_arrow_iterable(max_queue_size=2):
                    if batch.num_rows == 0:
                        continue
                    table = pa.Table.from_batches([batch])
                    for name in table.column_names:
                        if "properties" in name or "json" in name:
                            idx = table.schema.get_field_index(name)
                            table = table.set_column(idx, name,
table[name].cast(pa.string()))
                        if writer is None:
                            writer = pq.ParquetWriter(p["file"], table.schema)
                        writer.write_table(table)
                        rows += batch.num_rows
                    if writer:
                        writer.close()
                print(f"    Written {rows} rows -> {p['file'].name}")
            except Exception as e:
                print(f"BQ error {p['file'].name}: {e}")

        def merge_and_clean(self):
            if not self.marketing_path.exists():
                print("Marketing file missing.")
                return
            product_files = sorted(glob.glob(str(DATA_DIR /
"full_product_export_*.parquet")))
            if not product_files:
                print("No product files found.")
                return

            temp_path = DATA_DIR / "temp_clean_all_events.parquet"
            target_ts = pa.timestamp("us", tz="UTC")

```

```

writer = None
total = 0
for f in product_files:
    try:
        pf = pq.ParquetFile(f)
        for batch in pf.iter_batches(batch_size=65536):
            if batch.num_rows == 0:
                continue
            table = pa.Table.from_batches([batch])
            cols, names = [], []
            for name in table.column_names:
                col = table[name]
                if "properties" in name or "json" in name:
                    col = col.cast(pa.string())
                elif name == "event_time" and col.type != target_ts:
                    try:
                        col = col.cast(target_ts)
                    except Exception:
                        col = col.cast(target_ts, safe=False)
                cols.append(col)
                names.append(name)
            clean = pa.Table.from_arrays(cols, names=names)
            if writer is None:
                writer = pq.ParquetWriter(temp_path, clean.schema)
            writer.write_table(clean)
            total += batch.num_rows
        gc.collect()
    except Exception as e:
        print(f"Error {os.path.basename(f)}: {e}")
if writer:
    writer.close()
else:
    print("No data written.")
    return

df_marketing = pl.read_parquet(self.marketing_path).with_columns([
    pl.col("user_id").cast(pl.Int64),
    pl.col("original_purchase_date").cast(pl.Datetime("us")),
    pl.col("install_time").cast(pl.Datetime("us")),
]).unique(subset=["user_id"])

df_cities = None
if cities_csv_path.exists():
    df_cities = (pl.read_csv(cities_csv_path).select("city")
                  .with_columns(pl.lit(1,
dtype=pl.UInt8).alias("is_big_city_marker")).unique())

self.merged_path.parent.mkdir(parents=True, exist_ok=True)
writer = None
saved = 0
for batch in
pq.ParquetFile(temp_path).iter_batches(batch_size=5_000_000):
    df = pl.from_arrow(batch).with_columns([
        pl.col("user_id").cast(pl.Int64, strict=False),
        pl.col("event_time").cast(pl.Datetime("us")),
    ]).filter(pl.col("user_id").is_not_null())
    if df.height == 0:
        continue
    df = df.join(df_marketing, on="user_id", how="inner")
    if df.height == 0:
        continue
    df = df.filter(
        (pl.col("event_time") >= pl.col("install_time")) &

```

```

        (pl.col("event_time") <= pl.col("original_purchase_date") +
pl.duration(hours=trial_window_hours))
    )
    if df_cities is not None:
        df = (df.join(df_cities, on="city", how="left")
              .with_columns(pl.col("is_big_city_marker").fill_null(0)
              .alias("is_big_city"))
              .drop("is_big_city_marker"))
    else:
        df = df.with_columns(pl.lit(0,
dtype=pl.UInt8).alias("is_big_city"))
        df = df.unique(subset=["user_id", "event_time", "event_type"])
        if df.height > 0:
            table = df.to_arrow()
            if writer is None:
                writer = pq.ParquetWriter(self.merged_path, table.schema)
            writer.write_table(table)
            saved += df.height
        del df, batch
        gc.collect()
    if writer:
        writer.close()
    if temp_path.exists():
        temp_path.unlink()
    print(f"Merged {saved} rows -> {self.merged_path}")

def run_pipeline(self, start_date: datetime, final_end_date: datetime,
split_date: datetime):
    self.extract_product_data(start_date, final_end_date, split_date)
    self.merge_and_clean()

# FEATURE TRANSFORMERS
def _cohort_join(lf_user: pl.LazyFrame, lf_stats: pl.LazyFrame,
                 windows: list[tuple], stat_schema: list[str],
                 pre_trial_h: int = pre_trial_recent_hours) -> pl.LazyFrame:
    """Shared logic: join rolling stats and compute ratios for given time
    windows."""
    lf_user = lf_user.with_columns([
        pl.col("install_time").cast(pl.Date).alias("key_date_inst"),
        (pl.col("original_purchase_date") -
pl.duration(hours=pre_trial_h)).cast(pl.Date).alias("key_date_pretrial"),
pl.col("original_purchase_date").cast(pl.Date).alias("key_date_trial"),
        pl.col("country").fill_null("unknown"),
        pl.col("media_source").fill_null("unknown"),
    ])
    stat_date_col = "original_purchase_date" if "original_purchase_date" in
stat_schema else "cohort_date"
    lf_s = lf_stats.with_columns([
        pl.col(stat_date_col).cast(pl.Date).alias("stat_date"),
        pl.col("country").fill_null("unknown"),
        pl.col("media_source").fill_null("unknown"),
    ])
    updated_stat_schema = lf_s.collect_schema().names()
    user_schema = lf_user.collect_schema().names()

    for window_name, date_key in windows:
        target_cols = [c for c in user_schema if
c.startswith(f"{window_name}_")]
        pairs = []
        for uc in target_cols:

```

```

        for cand in [f"avg30d_{uc}", f"avg7d_{uc}", f"roll_avg_{uc}",
f"avg_{uc}"]:
            if cand in updated_stat_schema:
                pairs.append((uc, cand))
                break
        if not pairs:
            continue
        sel = ["stat_date", "country", "media_source"]
        rmap = {"stat_date": date_key}
        for _, sc in pairs:
            if sc not in sel:
                sel.append(sc)
                rmap[sc] = f"tmp_{window_name}_{sc}"
        curr = lf_s.select(sel).rename(rmap)
        lf_user = lf_user.join(curr, on=[date_key, "country",
"media_source"], how="left")
        ratio_exprs, drop_cols = [], []
        for uc, sc in pairs:
            tmp = f"tmp_{window_name}_{sc}"
            ratio_exprs.append((pl.col(uc) / (pl.col(tmp).fill_null(0) +
0.0001)).fill_null(0).alias(f"ratio_{uc}"))
            drop_cols.append(tmp)
        lf_user = lf_user.with_columns(ratio_exprs).drop(drop_cols)

    return lf_user.drop(["key_date_inst", "key_date_pretrial",
"key_date_trial"])

class StaticTransformer:
    def __init__(self):
        self.top_n_maps: dict = {}
        self.freq_maps: dict = {}
        self.is_fitted = False

    def calculate_cohort_stats(self, lf: pl.LazyFrame):
        if isinstance(lf, pl.DataFrame):
            lf = lf.lazy()
            schema = lf.collect_schema().names()
            cast_exprs = [pl.col(c).cast(pl.Float64, strict=False) for c in
num_cols if c in schema]
            if cast_exprs:
                lf = lf.with_columns(cast_exprs)

            date_c = "install_time" if "install_time" in schema else
"original_purchase_date"
            cols = list(dict.fromkeys([c for c in cat_cols + ["spend_usd",
date_c] if c in schema]))
            if cols:
                df = lf.select(cols).collect()
                max_date = df[date_c].max() if date_c in df.columns else None
                df_recent = df.filter(pl.col(date_c) >= max_date -
timedelta(days=30)) if max_date else df

                for col, n in top_n_config.items():
                    if col in df_recent.columns:
                        agg_col = "spend_usd" if "spend_usd" in df_recent.columns
                    else col
                        top = (df_recent.group_by(col)
                            .agg((pl.col(agg_col).sum() if "spend_usd" in
df_recent.columns else pl.len()).alias("s")))
                            .sort("s",
descending=True).head(n).get_column(col).to_list())
                        self.top_n_maps[col] = [str(x) for x in top if x is not
None]

```

```

        for col in cat_cols:
            if col in df.columns:
                counts = df[col].value_counts()
                self.freq_maps[col] =
dict(zip(counts[col].cast(pl.String).to_list(),
strict=False))
                counts["count"].to_list(),
            self.save()

        miss = [k for k in cohort_cols if k not in schema]
        if miss or not any(c in schema for c in num_cols):
            self._save_dummy_stats()
            return None
        cols_to_agg = [c for c in num_cols if c in schema]
        return
        (lf.with_columns(pl.col("original_purchase_date").cast(pl.Date))
            .group_by(cohort_cols)
            .agg([pl.col(c).mean().alias(f"avg_{c}") for c in
cols_to_agg]))

    def transform(self, lf: pl.LazyFrame, lf_stats_rolling: pl.LazyFrame =
None) -> pl.LazyFrame:
        if isinstance(lf, pl.DataFrame):
            lf = lf.lazy()
        if not self.is_fitted:
            self.load()
        schema = lf.collect_schema().names()
        meta = ["user_id", "original_purchase_date", "install_time",
"country", "media_source"]
        avail = [c for c in list(dict.fromkeys(meta + cat_cols + num_cols))
if c in schema]
        lf_s = lf.select(avail).group_by(user_col).first()

        fill = ([pl.col(c).cast(pl.Float64, strict=False).fill_null(0) for c
in num_cols if c in avail] +
                [pl.col(c).cast(pl.String).fill_null("unknown") for c in
cat_cols if c in avail])
        if fill:
            lf_s = lf_s.with_columns(fill)

        mask =
[pl.when(pl.col(c).is_in(top)).then(pl.col(c)).otherwise(pl.lit("Other")).ali
as(c)
            for c, top in self.top_n_maps.items() if c in avail]
        if mask:
            lf_s = lf_s.with_columns(mask)

        curr_cols = lf_s.collect_schema().names()
        inter = [pl.concat_str([pl.col(a), pl.col(b)],
separator="_").alias(f"inter_{a}_{b}")
            for a, b in interaction_pairs if a in curr_cols and b in
curr_cols]
        if inter:
            lf_s = lf_s.with_columns(inter)

        freq = [pl.col(c).cast(pl.String).replace(self.freq_maps[c],
default=None)
            .cast(pl.Int64, strict=False).fill_null(1).alias(f"freq_{c}")
            for c in cat_cols if c in self.freq_maps and c in curr_cols]
        if freq:
            lf_s = lf_s.with_columns(freq)

        if lf_stats_rolling is None:

```

```

        return lf_s

    stat_schema = lf_stats_rolling.collect_schema().names()
    lf_s = _cohort_join(lf_s, lf_stats_rolling, [{"pretrial",
"key_date_pretrial"}], stat_schema)

    final_cols = lf_s.collect_schema().names()
    cat_cast = [pl.col(c).cast(pl.Categorical)
                  for c in final_cols if c.startswith("inter_") or c in
cat_cols]
    if cat_cast:
        lf_s = lf_s.with_columns(cat_cast)
    return lf_s

    def save(self):
        static_artifacts_path.parent.mkdir(parents=True, exist_ok=True)
        with open(static_artifacts_path, "w") as f:
            json.dump({"top_n_maps": self.top_n_maps, "freq_maps":
self.freq_maps}, f)

    def load(self):
        if static_artifacts_path.exists():
            with open(static_artifacts_path) as f:
                data = json.load(f)
            self.top_n_maps = data.get("top_n_maps", {})
            self.freq_maps = data.get("freq_maps", {})
            self.is_fitted = True

    def _save_dummy_stats(self):
        dummy = pl.DataFrame({"original_purchase_date": [], "country": [],
"media_source": [], "avg_dummy": []},
                             schema={"original_purchase_date": pl.Date,
"country": pl.String,
                                     "media_source": pl.String, "avg_dummy":
pl.Float64})
        static_cohort_stats_path.parent.mkdir(parents=True, exist_ok=True)
        dummy.write_parquet(static_cohort_stats_path)

class TimeFeaturesTransformer:
    def __init__(self):
        self.trial_window_hours = trial_window_hours
        self.pre_trial_recent_hours = pre_trial_recent_hours
        self.target_events = target_events

    def calculate_cohort_stats(self, lf: pl.LazyFrame) -> pl.LazyFrame |
None:
        lf_user = self._base_features(lf)
        schema = lf_user.collect_schema().names()
        metrics = [c for c in schema if c.startswith(("pretrial_", "trial_"))
                    and c not in cohort_cols and c != user_col]
        if not metrics:
            return None
        return
(lf_user.with_columns(pl.col("original_purchase_date").cast(pl.Date))
          .group_by(cohort_cols)
          .agg([pl.col(m).mean().alias(f"avg_{m}") for m in metrics]))

    def transform(self, lf: pl.LazyFrame, lf_stats_rolling: pl.LazyFrame) ->
pl.LazyFrame:
        lf_user = self._base_features(lf)
        if lf_stats_rolling is None:
            return lf_user
        stat_schema = lf_stats_rolling.collect_schema().names()

```

```

        return _cohort_join(lf_user, lf_stats_rolling,
                             [("pretrial", "key_date_pretrial"), ("trial",
                             "key_date_trial")],
                             stat_schema, self.pre_trial_recent_hours)

    def _base_features(self, lf: pl.LazyFrame) -> pl.LazyFrame:
        all_events = sorted(self.target_events)
        lf = lf.sort([user_col, event_time_col])
        lf_prep = lf.with_columns([
            (pl.col(event_time_col) -
             pl.col("original_purchase_date")).dt.total_seconds().alias("diff"),
            np.sin(2 * np.pi * pl.col(event_time_col).dt.hour() /
24).alias("time_sin"),
            np.cos(2 * np.pi * pl.col(event_time_col).dt.hour() /
24).alias("time_cos"),
        ])
        pretrial_sec = self.pre_trial_recent_hours * 3600
        trial_sec = self.trial_window_hours * 3600
        pretrial_cond = (pl.col("diff") >= -pretrial_sec) & (pl.col("diff") <
0)
        trial_cond = (pl.col("diff") >= 0) & (pl.col("diff") <= trial_sec)

        schema = lf.collect_schema().names()
        agg =
[pl.first("original_purchase_date").cast(pl.Date).alias("original_purchase_da
te")]
        for col in ["country", "media_source", "install_time"]:
            if col in schema:
                agg.append(pl.first(col))
            elif col != "install_time":
                agg.append(pl.lit("unknown").alias(col))
            if "install_time" in schema:
                agg.append((pl.first("original_purchase_date") -
pl.first("install_time"))
                           .dt.total_seconds().fill_null(0).alias("ctx_install_to
_trial_sec"))

        agg.append(pl.first("original_purchase_date").dt.hour().alias("ctx_trial_star
t_hour"))

        for prefix, cond in [("pretrial", pretrial_cond), ("trial",
trial_cond)]:
            agg.extend(self._stats_exprs(prefix, cond))
            agg.extend(self._event_exprs(prefix, cond, all_events))
            agg.append(self._entropy_expr(prefix, cond))

        return lf_prep.group_by(user_col).agg(agg).fill_null(0).fill_nan(0)

    def _stats_exprs(self, prefix, cond):
        ft = pl.col("diff").filter(cond)
        return [
            pl.col("event_type").filter(cond).len().alias(f"{prefix}_count"),
            (ft.max() -
             ft.min()).fill_null(0).alias(f"{prefix}_duration_sec"),
            (pl.col("event_type").filter(cond).len() / ((ft.max() - ft.min())
+ 1)).alias(f"{prefix}_velocity"),
            ft.diff().mean().fill_null(0).alias(f"{prefix}_gap_mean"),
        ]

    def _event_exprs(self, prefix, cond, events):
        exprs = []
        offset = self.pre_trial_recent_hours * 3600 if prefix == "pretrial"
else 0
        for ev in events:

```

```

        safe = str(ev).replace(" ", "_").lower()[:40]
        ec = cond & (pl.col("event_type") == ev)
        time_expr = pl.col("diff") + offset if prefix == "pretrial" else
pl.col("diff")
        exprs += [

pl.col("event_type").filter(ec).len().alias(f"{prefix}_cnt_{safe}"),

pl.col("time_sin").filter(ec).mean().fill_null(0).alias(f"{prefix}_sin_{safe}"),

pl.col("time_cos").filter(ec).mean().fill_null(0).alias(f"{prefix}_cos_{safe}"),

        time_expr.filter(ec).min().fill_null(-
1).alias(f"{prefix}_sec_from_start_{safe}"),
        ]
    return exprs

    def _entropy_expr(self, prefix, cond):
        return (pl.col("event_type").filter(cond).value_counts(sort=False)
                .struct.field("count").entropy(normalize=True)
                .fill_nan(0).fill_null(0).alias(f"{prefix}_shannon_entropy"))

class TfIdfTransformer:
    def __init__(self):
        self.top_k = top_k
        self.vocab: dict = {}
        self.mapping_df = pl.DataFrame(EVENT_MAPPING_DATA)
        self.target_events = set(target_events)
        self.forbidden_events = set(forbidden_event_cats)

    def update_vocab(self, lf: pl.LazyFrame):
        bounds =
lf.select([pl.col("original_purchase_date").min().alias("mn"),

pl.col("original_purchase_date").max().alias("mx")]).collect()
        min_date, max_date = bounds["mn"][0], bounds["mx"][0]
        if min_date is None:
            self.vocab = {}
            return
        global_terms: dict = {}
        total_users = 0
        current = min_date.replace(day=1)
        while current <= max_date:
            nxt = (current + timedelta(days=32)).replace(day=1)
            chunk = lf.filter((pl.col("original_purchase_date") >= current) &
                             (pl.col("original_purchase_date") < nxt))
            try:
                df = self._base_features(chunk,
fit_mode=True).collect(streaming=True)
                if df.height > 0:
                    total_users += df.height
                    for col in [c for c in df.columns if "_cnt_" in c]:
                        term = col.split("_cnt_", 1)[1]
                        global_terms[term] = global_terms.get(term, 0) +
(df[col].sum() or 0)
            except Exception as e:
                print(f"Chunk {current}: {e}")
            del chunk
            gc.collect()
            current = nxt

```

```

        sorted_terms = sorted(global_terms.items(), key=lambda x: x[1],
reverse=True)[:self.top_k]
        self.vocab = {t: math.log((total_users + 1) / (min(cnt, total_users)
+ 1)) + 1
                        for t, cnt in sorted_terms}
        self._save_vocab()

    def calculate_cohort_stats(self, lf: pl.LazyFrame) -> pl.LazyFrame:
        lf_user = self._user_features(lf)
        schema = lf_user.collect_schema().names()
        cols = [c for c in schema if any(x in c for x in ["_tfidf_", "_dur_",
"_is_conv_", "_rate_"])]
        if not cols:
            return lf_user.select(cohort_cols).unique()
        return
        (lf_user.with_columns(pl.col("original_purchase_date").cast(pl.Date))
         .group_by(cohort_cols)
         .agg([pl.col(c).cast(pl.Float64).mean().alias(f"avg_{c}") for
c in cols]))

    def transform(self, lf: pl.LazyFrame, lf_stats_rolling: pl.LazyFrame) ->
pl.LazyFrame:
        if isinstance(lf, pl.DataFrame):
            lf = lf.lazy()
        if not self.vocab:
            self._load_vocab()
        lf_user = self._user_features(lf)
        if lf_stats_rolling is None:
            return lf_user
        stat_schema = lf_stats_rolling.collect_schema().names()
        return _cohort_join(lf_user, lf_stats_rolling,
                            [("pretrial", "key_date_pretrial"), ("trial",
"key_date_trial")],
                            stat_schema)

    def _user_features(self, lf: pl.LazyFrame) -> pl.LazyFrame:
        lf_base = self._base_features(lf, fit_mode=False)
        cols = lf_base.collect_schema().names()
        exprs = []
        for prefix in ["pretrial", "trial"]:
            for term, idf in self.vocab.items():
                cnt_col = f"{prefix}_cnt_{term}"
                exprs.append((pl.col(cnt_col) *
idf).alias(f"{prefix}_tfidf_{term}")
                            if cnt_col in cols else
pl.lit(0.0).alias(f"{prefix}_tfidf_{term}"))
        return lf_base.with_columns(exprs)

    def _enrich(self, lf: pl.LazyFrame) -> pl.LazyFrame:
        lf = lf.join(self.mapping_df.lazy(), on="event_type", how="left")
        tevts = list(self.target_events)
        lf = lf.with_columns([
            pl.when(pl.col("category").is_null() &
pl.col("event_type").is_in(tevts))
                .then(pl.lit("other")).otherwise(pl.col("category")).alias("cat
egory"),
            pl.when(pl.col("sentiment").is_null() &
pl.col("event_type").is_in(tevts))
                .then(pl.lit("neutral")).otherwise(pl.col("sentiment")).alias("
sentiment"),
        ])
        if self.forbidden_events:
            lf =
lf.filter(~pl.col("event_type").is_in(list(self.forbidden_events)))

```

```

        lf =
lf.filter(~pl.col("category").is_in(list(self.forbidden_events)))
        return lf.with_columns([pl.col("category").fill_null("unknown"),
                                pl.col("sentiment").fill_null("unknown")])

    def _base_features(self, lf: pl.LazyFrame, fit_mode: bool = False) ->
pl.LazyFrame:
        if user_col not in lf.collect_schema().names():
            return pl.DataFrame({user_col: []}, schema={user_col:
pl.Int64}).lazy()
        lf = self._enrich(lf).sort([user_col, event_time_col])
        lf = lf.with_columns(
            (pl.col(event_time_col) -
pl.col("original_purchase_date")).dt.total_seconds().alias("diff"))
        ptrial = pre_trial_recent_hours * 3600
        twindow = trial_window_hours * 3600
        w_pre = (pl.col("diff") >= -ptrial) & (pl.col("diff") < 0)
        w_tri = (pl.col("diff") >= 0) & (pl.col("diff") <= twindow)
        windows = [("pretrial", w_pre), ("trial", w_tri)]

        lf_meta = lf.group_by(user_col).agg([
            pl.first("original_purchase_date").cast(pl.Date),
            pl.first("country"), pl.first("media_source"),
pl.first("install_time"),
        ])
        target_terms = list(self.vocab.keys()) if self.vocab and not fit_mode
    else []
        final_dfs, conv_dfs = [], []

        unique_ev = {e for pair in conversion_pairs for e in pair}
        for prefix, cond in windows:
            lf_win = lf.filter(cond)
            s1 = lf_win.select([user_col,
pl.col("event_type").alias("term")])
            s2 = lf_win.select([user_col, (pl.lit("cat_") +
pl.col("category")).alias("term")])
            s3 = lf_win.select([user_col, (pl.lit("sent_") +
pl.col("sentiment")).alias("term")])
            s4 = (lf_win.with_columns(
                (pl.col("event_type") + "_" +
pl.col("event_type").shift(-1).over(user_col)).alias("term"))
                .filter(pl.col("term").is_not_null()).select([user_col,
"term"]))
            all_terms = pl.concat([s1, s2, s3, s4])

            if fit_mode:
                top150 = (all_terms.group_by("term").len().sort("len",
descending=True)
                    .head(150).collect().get_column("term").to_list())
                exprs = [(pl.col("term") ==
t).sum().alias(f"{prefix}_cnt_{t}") for t in top150]
                if exprs:
                    final_dfs.append(all_terms.group_by(user_col).agg(exprs))
            else:
                exprs = [(pl.col("term") ==
t).sum().alias(f"{prefix}_cnt_{t}") for t in target_terms]
                if exprs:
                    final_dfs.append(all_terms.group_by(user_col).agg(exprs))

        cat_agg = []
        for cat in ALL_CATEGORIES:
            cat_agg += [
                pl.col("event_type").filter(pl.col("category") ==
cat).count().alias(f"tmp_{prefix}_{cat}_total"),

```

```

        pl.col("event_type").filter((pl.col("category") == cat) &
(pl.col("sentiment") == "positive"))
        .count().alias(f"tmp_{prefix}_{cat}_pos"),
        pl.col("event_type").filter((pl.col("category") == cat) &
(pl.col("sentiment") == "negative"))
        .count().alias(f"tmp_{prefix}_{cat}_neg"),
    ]
    if cat_agg:
        lf_cat = lf_win.group_by(user_col).agg(cat_agg)
        ratio_exprs = []
        drop_cols = []
        for cat in ALL_CATEGORIES:
            tot, pos, neg = f"tmp_{prefix}_{cat}_total",
f"tmp_{prefix}_{cat}_pos", f"tmp_{prefix}_{cat}_neg"
            ratio_exprs += [
                (pl.col(pos) / (pl.col(tot) +
0.001)).fill_null(0).alias(f"{prefix}_rate_{cat}_pos"),
                (pl.col(neg) / (pl.col(tot) +
0.001)).fill_null(0).alias(f"{prefix}_rate_{cat}_neg"),
            ]
            drop_cols += [tot, pos, neg]

final_dfs.append(lf_cat.with_columns(ratio_exprs).drop(drop_cols))

        time_agg = [pl.col("event_time").filter(cond &
(pl.col("event_type") == ev)).min().alias(f"time_{ev}")
            for ev in unique_ev]
        lf_times = lf.group_by(user_col).agg(time_agg)
        conv_exprs = []
        for s_ev, e_ev in conversion_pairs:
            sc, ec = f"time_{s_ev}", f"time_{e_ev}"
            pair = f"{s_ev}_to_{e_ev}"
            dur = (pl.col(ec) - pl.col(sc)).dt.total_seconds()
            ok = (pl.col(sc).is_not_null()) & (pl.col(ec).is_not_null())
& (dur >= 0)
            conv_exprs += [
                pl.when(ok).then(dur).otherwise(0).fill_null(0).alias(f"{prefix}_dur_{pair}")
            ],
            pl.when(ok).then(1).otherwise(0).alias(f"{prefix}_is_conv_{pair}"),
        ]
        if conv_exprs:
            conv_dfs.append(lf_times.select([user_col] + conv_exprs))

    lf_final = lf_meta
    for df in final_dfs + conv_dfs:
        lf_final = lf_final.join(df, on=user_col, how="left")
    return lf_final.fill_null(0)

def _save_vocab(self):
    tfidf_vocab_path.parent.mkdir(parents=True, exist_ok=True)
    with open(tfidf_vocab_path, "w") as f:
        json.dump(self.vocab, f)

def _load_vocab(self):
    if tfidf_vocab_path.exists():
        with open(tfidf_vocab_path) as f:
            self.vocab = json.load(f)

```

```
# UNIFIED STATS REGISTRY
```

```

class UnifiedStatsRegistry:
    def __init__(self, stats_path: Path):
        self.stats_path = Path(stats_path)

    def update_daily_stats(self, list_of_lfs: list):
        processed = [item.lazy() if isinstance(item, pl.DataFrame) else item
                      for item in list_of_lfs if item is not None]
        if not processed:
            return
        lf_master = processed[0]
        for lf in processed[1:]:
            lf_master = lf_master.join(lf, on=cohort_cols, how="full",
coalesce=True)
        try:
            df = lf_master.collect()
            if df.height == 0:
                return
            df =
df.with_columns([pl.col("original_purchase_date").cast(pl.Date),
pl.col("country").cast(pl.String).fill_null("unknown"),
pl.col("media_source").cast(pl.String).fill_null("unknown")])
            metric_cols = [c for c in df.columns if c.startswith("avg_")]
            if metric_cols:
                df_broad = (df.lazy().group_by(["original_purchase_date",
"country"]))
                                .agg([pl.col(c).mean().alias(f"{c}_broad") for c
in metric_cols]).collect())
                df_global = (df.lazy().group_by(["original_purchase_date"])
                                .agg([pl.col(c).mean().alias(f"{c}_global") for
c in metric_cols]).collect())
                df = (df.lazy()
                                .join(df_broad.lazy(), on=["original_purchase_date",
"country"], how="left")
                                .join(df_global.lazy(), on=["original_purchase_date"],
how="left")
                                .collect())
        except Exception as e:
            print(f"Registry error: {e}")
            return

        if self.stats_path.exists():
            try:
                lf_hist = pl.scan_parquet(self.stats_path)
                new_dates = df["original_purchase_date"].unique().to_list()
                df =
pl.concat([lf_hist.filter(~pl.col("original_purchase_date").is_in(new_dates))
.collect(), df],
                                how="diagonal")
            except Exception as e:
                print(f"Overwriting registry: {e}")
                self.stats_path.parent.mkdir(parents=True, exist_ok=True)
                df.sort(cohort_cols).write_parquet(self.stats_path)
                print(f"Registry updated. Shape: {df.shape}")

    def get_rolling_stats(self, lookback_days: int = 30) -> pl.LazyFrame:
        if not self.stats_path.exists():
            return pl.DataFrame([]).lazy()
        lf = pl.scan_parquet(self.stats_path)
        try:
            max_date =
lf.select(pl.col("original_purchase_date").max()).collect().item()

```

```

        if max_date:
            lf = lf.filter(pl.col("original_purchase_date") >= max_date -
timedelta(days=65))
        except Exception:
            pass
        lf =
lf.sort(cohort_cols).with_columns([pl.col("country").fill_null("unknown"),
pl.col("media_source").fill_null("unknown")])
        all_cols = lf.collect_schema().names()
        base_metrics = [c for c in all_cols if c.startswith("avg_") and not
c.endswith(("_broad", "_global"))]
        if not base_metrics:
            return lf

        shift_exprs = []
        for col in base_metrics:
            clean = col.replace("avg_", "")
            shift_exprs.append(pl.col(col).shift(1).over(["country",
"media_source"]).alias(f"shift_{clean}"))
            if f"{col}_broad" in all_cols:

shift_exprs.append(pl.col(f"{col}_broad").shift(1).over(["country",
"media_source"])
                    .alias(f"shift_{clean}_broad"))
            if f"{col}_global" in all_cols:

shift_exprs.append(pl.col(f"{col}_global").shift(1).over(["country",
"media_source"])
                    .alias(f"shift_{clean}_global"))
        lf = lf.with_columns(shift_exprs)

        def roll(c, w):
            return pl.col(c).rolling_mean(window_size=w,
min_periods=1).over(["country", "media_source"])

        final_exprs = []
        for col in base_metrics:
            clean = col.replace("avg_", "")
            s1, s2, s3 = f"shift_{clean}", f"shift_{clean}_broad",
f"shift_{clean}_global"
            has2, has3 = f"{col}_broad" in all_cols, f"{col}_global" in
all_cols

            for days, alias in [(30, f"avg30d_{clean}"), (7,
f"avg7d_{clean}")]:
                cand = [pl.when(roll(s1, days) > 0).then(roll(s1,
days)).otherwise(None)]
                if has2:
                    cand.append(pl.when(roll(s2, days) > 0).then(roll(s2,
days)).otherwise(None))
                if has3:
                    cand.append(roll(s3, days))

            final_exprs.append(pl.coalesce(cand).fill_null(0).alias(alias))

        cand1d = [pl.col(s1)]
        if has2:
            cand1d.append(pl.col(s2))
        if has3:
            cand1d.append(pl.col(s3))

        final_exprs.append(pl.coalesce(cand1d).fill_null(0).alias(f"roll_avg_{clean}"))

```

```

        return lf.with_columns(final_exprs).select(cohort_cols +
[e.meta.output_name() for e in final_exprs])

# DAILY STATS UPDATE
TEMP_STATS_DIR = Path("data/temp_daily_stats")

def run_stats_update(target_date: date, source_mode: str = "file",
                    force_restart: bool = False, specific_transformer: str =
None):
    print(f"\nSTATUS UPDATE | target={target_date} | source={source_mode}")
    if TEMP_STATS_DIR.exists():
        shutil.rmtree(TEMP_STATS_DIR)
    TEMP_STATS_DIR.mkdir(parents=True, exist_ok=True)

    start_filter = None
    if force_restart and unified_stats_path.exists():
        unified_stats_path.unlink()
    elif not force_restart and unified_stats_path.exists():
        try:
            df_dates =
pl.scan_parquet(unified_stats_path).select("original_purchase_date").unique()
.collect()
            if df_dates.height > 0:
                start_filter = df_dates["original_purchase_date"].max() -
timedelta(days=1)
        except Exception:
            pass

    if source_mode == "file":
        lf_raw = fetch_full_history()
        if start_filter:
            lf_raw = lf_raw.filter(pl.col("original_purchase_date") >=
start_filter)
        lf_raw = lf_raw.filter(pl.col("original_purchase_date") <=
target_date)
    else:
        start = start_filter or date(2023, 6, 1)
        df_raw = fetch_full_user_data(datetime.combine(start,
datetime.min.time()),
datetime.combine(target_date,
datetime.max.time()), mode="train")
        if df_raw.is_empty():
            shutil.rmtree(TEMP_STATS_DIR)
            return
        lf_raw = df_raw.lazy()

    registry = UnifiedStatsRegistry(stats_path=unified_stats_path)
    all_transformers = [("TIME", TimeFeaturesTransformer()), ("STATIC",
StaticTransformer()),
                        ("TFIDF", TfIdfTransformer())]
    transformers = ([t for t in all_transformers if t[0] ==
specific_transformer.upper()]
                    if specific_transformer else all_transformers)

    for name, transformer in transformers:
        if name == "TFIDF":
            try:
                transformer.update_vocab(lf_raw)
            except Exception as e:
                print(f"Vocab update error: {e}")

```

```

        return
    gc.collect()

    all_merged = []
    for yr in range(2020, 2027):
        lf_year =
        lf_raw.filter(pl.col("original_purchase_date").is_between(date(yr, 1, 1),
        date(yr, 12, 31)))
        try:
            if lf_year.fetch(1).height == 0:
                continue
        except Exception:
            continue

        year_files = []
        for name, transformer in transformers:
            try:
                lf_stat = transformer.calculate_cohort_stats(lf_year)
                if lf_stat is None:
                    continue
                tmp = TEMP_STATS_DIR / f"{name.lower()}_stats_{yr}.parquet"
                lf_stat.sink_parquet(tmp)
                year_files.append(tmp)
                del lf_stat
            except Exception as e:
                print(f"{name} year {yr}: {e}")

        if year_files:
            lf_yr = pl.scan_parquet(str(year_files[0]))
            for f in year_files[1:]:
                lf_yr = lf_yr.join(pl.scan_parquet(str(f)), on=cohort_cols,
            how="full", coalesce=True)
            merged = TEMP_STATS_DIR / f"merged_{yr}.parquet"
            lf_yr.sink_parquet(merged)
            all_merged.append(merged)
            for f in year_files:
                f.unlink()

        if all_merged:
            lf_final = pl.concat([pl.scan_parquet(str(f)) for f in all_merged],
            how="diagonal")
            registry.update_daily_stats([lf_final])

    shutil.rmtree(TEMP_STATS_DIR)

# HISTORICAL DATASET BUILDER
STATS_PATH = Path("data/unified_stats_history_6h.parquet")
START_DATE = datetime(2023, 6, 1)
END_DATE = datetime(2025, 12, 31)
BUILD_DIR = Path("data/temp_build")
CHUNKS_DIR = BUILD_DIR / "chunks"
NUM_PARTITIONS = 10
BASE_FEATURE_TYPE = "time"
EXCLUDE_FROM_CLEANING = {"user_id", "is_converted", "target"}
SHARED_COLS = {"original_purchase_date", "is_converted", "event_time",
"install_time", "country", "media_source"}

TRANSFORMERS_CONFIG = {
    "time": TimeFeaturesTransformer(),
    "static": StaticTransformer(),
    "tfidf": TfIdfTransformer(),

```

```

}

def _enforce_schema(df: pl.DataFrame) -> pl.DataFrame:
    passthrough = {"user_id", "original_purchase_date", "event_time",
"install_time",
                    "country", "media_source", "is_converted"}
    exprs = []
    for col, dtype in df.schema.items():
        if col in passthrough:
            exprs.append(pl.col(col))
        elif dtype in [pl.String, pl.Categorical]:
            exprs.append(pl.col(col).cast(pl.String).fill_null("unknown"))
        elif dtype == pl.Float64:
            exprs.append(pl.col(col).cast(pl.Float32).fill_null(0.0))
        elif dtype == pl.Float32:
            exprs.append(pl.col(col).fill_null(0.0))
        elif dtype in [pl.Int64, pl.Int32]:
            exprs.append(pl.col(col).cast(pl.Int32,
strict=False).fill_null(0))
        elif dtype == pl.Boolean:
            exprs.append(pl.col(col).fill_null(False).cast(pl.Int8))
        else:
            exprs.append(pl.col(col))
    return df.select(exprs)

def build_feature_set(feature_type: str, transformer) -> Path | None:
    print(f"\n{'='*40}\nGENERATING: {feature_type.upper()}\n{'='*40}")
    if BUILD_DIR.exists():
        shutil.rmtree(BUILD_DIR)
    CHUNKS_DIR.mkdir(parents=True, exist_ok=True)

    registry = UnifiedStatsRegistry(stats_path=STATS_PATH)
    lf_roll = registry.get_rolling_stats(lookback_days=30).filter(
        pl.col("original_purchase_date").is_between(START_DATE -
timedelta(days=40), END_DATE + timedelta(days=5)))

    meta_drop = ["original_purchase_date", "country", "media_source",
"event_time", "install_time"]
    lf_history = fetch_full_history()
    current, chunk_id = START_DATE, 0

    while current <= END_DATE:
        nxt = current + relativedelta(months=1)
        month_str = current.strftime("%Y-%m")
        lf_month = (lf_history
                    .filter(pl.col("original_purchase_date").is_between(curre
nt, nxt - timedelta(days=1)))
                    .filter(pl.col("event_time") <=
pl.col("original_purchase_date") + pl.duration(hours=trial_window_hours)))
        df_stats =
lf_roll.filter(pl.col("original_purchase_date").is_between(
current - timedelta(days=35), nxt)).collect()

        for part in range(NUM_PARTITIONS):
            try:
                df_part = lf_month.filter((pl.col("user_id") %
NUM_PARTITIONS) == part).collect()
                if df_part.height == 0:
                    continue
                keys = [c for c in ["user_id", "original_purchase_date",
"is_converted",

```

```

        "event_time", "install_time", "country",
        "media_source"]
        if c in df_part.columns]
        df_merged = df_part.select(keys).unique(subset=["user_id"],
keep="first")
        lf_feat = transformer.transform(df_part.lazy(),
df_stats.lazy())
        lf_feat = lf_feat.unique(subset=["user_id"], keep="first")
        drop_c = [c for c in meta_drop if c in
lf_feat.collect_schema().names() and c != "user_id"]
        if drop_c:
            lf_feat = lf_feat.drop(drop_c)
        df_merged = df_merged.join(lf_feat.collect(), on="user_id",
how="left")
        df_merged = _enforce_schema(df_merged)
        df_merged.write_parquet(CHUNKS_DIR /
f"chunk_{chunk_id:05d}_{month_str}_p{part}.parquet")
        chunk_id += 1
        del df_part, df_merged
        gc.collect()
    except Exception as e:
        print(f"Partition {part} error: {e}")
        gc.collect()
    del df_stats
    gc.collect()
    current = nxt

chunks = list(CHUNKS_DIR.glob("*.parquet"))
if not chunks:
    return None

ts = datetime.now().strftime("%d%m%H%M")
out = DATA_DIR / f"{ts}_{feature_type}_dataset_6h.parquet"
lf_raw = pl.scan_parquet(str(CHUNKS_DIR / "*.parquet"))
all_c = lf_raw.collect_schema().names()
mandatory = {"user_id", "original_purchase_date", "is_converted",
"event_time"}
lf_raw.select([c for c in all_c if c in mandatory or c not in
SHARED_COLS]).sink_parquet(out)
shutil.rmtree(BUILD_DIR)
return out

def merge_and_finalize(generated_files: dict):
    if BASE_FEATURE_TYPE not in generated_files:
        print(f"Base '{BASE_FEATURE_TYPE}' missing.")
        return
    lf = pl.scan_parquet(generated_files[BASE_FEATURE_TYPE])
    for ftype, fpath in generated_files.items():
        if ftype == BASE_FEATURE_TYPE:
            continue
        lf_next = pl.scan_parquet(fpath)
        next_cols = lf_next.collect_schema().names()
        sel = ["user_id"] + [c for c in next_cols if c not in SHARED_COLS and
c != "user_id"]
        lf = lf.join(lf_next.select(sel), on="user_id", how="left")

    final_cols = lf.collect_schema().names()
    right_dups = [c for c in final_cols if c.endswith("_right")]
    if right_dups:
        lf = lf.drop(right_dups)
    lf = lf.filter(pl.col("install_time") <=
pl.col("original_purchase_date"))

```

```

    schema = lf.collect_schema()
    num_c = [c for c, dt in schema.items()
              if c not in EXCLUDE_FROM_CLEANING and c not in SHARED_COLS
              and not (any(x in c.lower() for x in ["sin_", "cos_", "_sin",
              "_cos"])) and "ratio" not in c.lower())
              and dt in [pl.Float64, pl.Float32, pl.Int64, pl.Int32, pl.Int16,
              pl.Int8]]
    if num_c:
        p99 = lf.select([pl.col(c).quantile(0.99).alias(c) for c in
        num_c]).collect()
        exprs = []
        for c in num_c:
            thr = p99[c][0]
            expr = pl.col(c).clip(lower_bound=0)
            if thr is not None:
                expr = expr.clip(upper_bound=thr)
            exprs.append(expr.log1p().alias(c))
        lf = lf.with_columns(exprs)

    cols_chk = [c for c in lf.collect_schema().names()
                if c not in EXCLUDE_FROM_CLEANING and c not in SHARED_COLS]
    uniq = lf.select([pl.col(c).n_unique().alias(c) for c in
    cols_chk]).collect()
    const_cols = [c for c in cols_chk if uniq[c][0] <= 1]
    if const_cols:
        lf = lf.drop(const_cols)

    out = DATA_DIR /
    f"{datetime.now().strftime('%d%m%H%M')} _merged_full.parquet"
    lf.sink_parquet(out)
    print(f"Final dataset: {out}")

def build_main():
    generated = {}
    for name, transformer in TRANSFORMERS_CONFIG.items():
        try:
            path = build_feature_set(name, transformer)
            if path:
                generated[name] = path
        except Exception as e:
            print(f"FAIL {name}: {e}")
            traceback.print_exc()
    if generated:
        merge_and_finalize(generated)

# TRAINING UTILITIES (shared across XGBoost / LightGBM / CatBoost)
DATA_PATH = Path("data/03051524_merged_full.parquet")
TARGET_COL = "is_converted"
DATE_COL = "original_purchase_date"
ID_COL = "user_id"
TEST_SIZE = 0.2
TRAIN_DATE_MIN = datetime(2023, 6, 1)
TRAIN_DATE_MAX = datetime(2025, 11, 13)
TIMESTAMP = datetime.now().strftime("%d%m%H%M")

def calculate_ece(y_true, y_prob, n_bins=10):
    prob_true, prob_pred = calibration_curve(y_true, y_prob, n_bins=n_bins,
    strategy="uniform")
    edges = np.linspace(0, 1, n_bins + 1)
    ids = np.digitize(y_prob, edges) - 1

```

```

counts = np.bincount(ids, minlength=n_bins)
return np.sum(np.abs(prob_true - prob_pred) * (counts[counts > 0] /
len(y_prob)))

def calculate_psi(expected, actual, buckets=10):
    bp = np.percentile(expected, np.arange(0, buckets + 1) / buckets * 100)
    e_pct = np.histogram(expected, bp)[0] / len(expected)
    a_pct = np.histogram(actual, bp)[0] / len(actual)
    return sum((e - a) * np.log((e or 1e-4) / (a or 1e-4)) for e, a in
zip(e_pct, a_pct))

def load_and_prep_data(to_category: bool = True):
    lf = pl.scan_parquet(DATA_PATH)
    schema = lf.collect_schema()
    all_cols = schema.names()
    lf =
lf.filter(pl.col(DATE_COL).cast(pl.Datetime).is_between(TRAIN_DATE_MIN,
TRAIN_DATE_MAX))
    exclude = {TARGET_COL, DATE_COL, ID_COL, "event_time", "install_time",
"cohort_date", "month_year"}
    feature_cols, cat_feats, exprs = [], [], [
        pl.col(ID_COL), pl.col(DATE_COL).cast(pl.Datetime),
pl.col(TARGET_COL),
    ]
    for col in all_cols:
        if col in exclude:
            continue
        dt = schema[col]
        if dt in (pl.Date, pl.Datetime, pl.Duration):
            continue
        feature_cols.append(col)
        if dt in (pl.String, pl.Categorical, pl.Object, pl.Boolean):
            if dt == pl.Boolean:
                exprs.append(pl.col(col).fill_null(False).cast(pl.Int8))
            else:
                exprs.append(pl.col(col).fill_null("unknown").cast(pl.String))
                cat_feats.append(col)
        elif dt in (pl.Float64, pl.Float32):
            exprs.append(pl.col(col).fill_null(0.0).cast(pl.Float32))
        elif dt in (pl.Int64, pl.Int32):
            exprs.append(pl.col(col).fill_null(0).cast(pl.Int32))
        else:
            exprs.append(pl.col(col))
    df = lf.select(exprs).sort(DATE_COL).collect().to_pandas()
    if to_category:
        for col in cat_feats:
            df[col] = df[col].astype("category")
    return df, feature_cols, cat_feats

def get_holdout_split(df, feature_cols, test_size=TEST_SIZE):
    idx = int(len(df) * (1 - test_size))
    return (df.iloc[:idx][feature_cols], df.iloc[:idx][TARGET_COL],
df.iloc[idx:][feature_cols], df.iloc[idx:][TARGET_COL])

def calibrate_threshold(y_true, y_prob, model_name: str) -> float:
    prec, rec, thr = precision_recall_curve(y_true, y_prob)
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        f1 = np.nan_to_num(2 * prec * rec / (prec + rec))

```

```

    opt_idx = np.argmax(f1)
    opt_thr = thr[opt_idx] if opt_idx < len(thr) else 0.5
    plt.figure(figsize=(10, 6))
    plt.plot(thr, prec[:-1], "b--", linewidth=2, label="Precision")
    plt.plot(thr, rec[:-1], "g-", linewidth=2, label="Recall")
    plt.plot(thr, f1[:-1], "r-", linewidth=3, label="F1-Score")
    plt.axvline(x=opt_thr, color="k", linestyle=":", label=f"Optimal:
{opt_thr:.2f}")
    plt.title(f"Threshold Calibration for CAPI ({model_name.upper()}"),
    fontsize=14)
    plt.xlabel("Threshold"); plt.ylabel("Metric value")
    plt.xlim([0, 1]); plt.ylim([0, 1.05]); plt.legend(); plt.grid(alpha=0.4);
plt.tight_layout()
    plt.savefig(REPORTS_DIR /
f"{TIMESTAMP}_{model_name}_threshold_calibration.png"); plt.close()
    return opt_thr

def evaluate_and_save(model, X_test, y_test, feature_cols, perf_metrics,
model_name: str):
    sample_size = min(1000, len(X_test))
    t0 = time.time()
    _ = model.predict_proba(X_test.iloc[:sample_size])
    inf_ms = (time.time() - t0) * 1000 * (1000 / sample_size)

    y_prob = model.predict_proba(X_test)[: , 1]
    thr = calibrate_threshold(y_test, y_prob, model_name)
    y_bin = (y_prob > thr).astype(int)

    auc = roc_auc_score(y_test, y_prob)
    ap = average_precision_score(y_test, y_prob)
    acc = accuracy_score(y_test, y_bin)
    ll = log_loss(y_test, y_prob)
    mcc = matthews_corrcoef(y_test, y_bin)
    brier = brier_score_loss(y_test, y_prob)
    ece = calculate_ece(y_test, y_prob)
    prec = precision_score(y_test, y_bin)
    rec = recall_score(y_test, y_bin)
    f1 = f1_score(y_test, y_bin)

    print(f"\n{model_name.upper()} | AUC={auc:.4f} | PR-AUC={ap:.4f} |
F1={f1:.4f} | "
        f"Prec={prec:.4f} | Rec={rec:.4f} | MCC={mcc:.4f} | ECE={ece:.4f}")
    print(f"Train: {perf_metrics['training_time_sec']:.1f}s | "
        f"Inference/1k: {inf_ms:.1f}ms | RAM:
{perf_metrics['process_ram_increase_mb']:.0f}MB")
    print(classification_report(y_test, y_bin))

    metrics = {
        "business_metrics": {"precision": float(prec), "recall": float(rec),
"fl_score": float(f1),
                                "roc_auc": float(auc), "pr_auc": float(ap),
"decision_threshold": float(thr)},
        "technical_metrics": {"mcc": float(mcc), "brier_score": float(brier),
"ece": float(ece),
                                "accuracy": float(acc), "r2_score":
float(r2_score(y_test, y_prob)),
                                "mse": float(mean_squared_error(y_test,
y_prob)), "log_loss": float(ll)},
        "performance_metrics": {"training_time_sec":
float(perf_metrics["training_time_sec"]),
                                "inference_time_per_1000_ms": float(inf_ms),
                                "process_ram_increase_mb":
float(perf_metrics["process_ram_increase_mb"])},

```

```

    }
    with open(REPORTS_DIR / f"{TIMESTAMP}_{model_name}_metrics.json", "w") as
f:
        json.dump(metrics, f, indent=4)

    cm = confusion_matrix(y_test, y_bin)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Greens")
    plt.title(f"Confusion Matrix {model_name} (thr={thr:.2f})")
    plt.xlabel("Predicted"); plt.ylabel("True")
    plt.savefig(REPORTS_DIR /
f"{TIMESTAMP}_{model_name}_confusion_matrix.png"); plt.close()

    prob_true, prob_pred = calibration_curve(y_test, y_prob, n_bins=10)
    plt.figure(figsize=(6, 6))
    plt.plot(prob_pred, prob_true, marker="o", label=model_name)
    plt.plot([0, 1], [0, 1], "--", label="Perfect")
    plt.title(f"Calibration Curve (ECE={ece:.3f})")
    plt.xlabel("Mean Predicted Prob"); plt.ylabel("Fraction Positives");
plt.legend()
    plt.savefig(REPORTS_DIR /
f"{TIMESTAMP}_{model_name}_calibration_curve.png"); plt.close()

    imp = (model.feature_importances_ if hasattr(model,
"feature_importances_")
           else model.get_feature_importance(type="PredictionValuesChange"))
    imp_df = pd.DataFrame({"Feature": feature_cols, "Importance":
imp}).sort_values("Importance", ascending=False)
    imp_df.to_csv(REPORTS_DIR / f"{TIMESTAMP}_{model_name}_importance.csv",
index=False)
    plt.figure(figsize=(10, 12))
    sns.barplot(data=imp_df.head(40), x="Importance", y="Feature",
hue="Feature", legend=False, palette="viridis")
    plt.title(f"Top 40 Features ({model_name})")
    plt.tight_layout()
    plt.savefig(REPORTS_DIR /
f"{TIMESTAMP}_{model_name}_feature_importance.png"); plt.close()

def generate_shap_plots(model, X_test, model_name: str):
    explainer = shap.TreeExplainer(model)
    X_sample = X_test.iloc[:1000]
    shap_vals = explainer(X_sample, check_additivity=False)
    if len(shap_vals.shape) == 3:
        shap_vals = shap_vals[:, :, 1]
    plt.figure(figsize=(12, 8))
    shap.plots.beeswarm(shap_vals, max_display=20, show=False)
    plt.title(f"SHAP Summary ({model_name})", fontsize=14)
    plt.tight_layout()
    plt.savefig(REPORTS_DIR / f"{TIMESTAMP}_{model_name}_shap_summary.png",
bbox_inches="tight"); plt.close()
    plt.figure(figsize=(10, 8))
    shap.plots.waterfall(shap_vals[0], max_display=15, show=False)
    plt.title(f"SHAP Waterfall ({model_name})", fontsize=14)
    plt.tight_layout()
    plt.savefig(REPORTS_DIR / f"{TIMESTAMP}_{model_name}_shap_waterfall.png",
bbox_inches="tight"); plt.close()

def _measure_fit(model, fit_fn) -> tuple:
    tracemalloc.start()
    proc = psutil.Process(os.getpid())
    ram_before = proc.memory_info().rss / 1e6
    t0 = time.time()

```

```

fit_fn(model)
elapsed = time.time() - t0
_, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
ram_delta = proc.memory_info().rss / 1e6 - ram_before
return {"training_time_sec": elapsed, "peak_python_ram_mb": peak / 1e6,
        "process_ram_increase_mb": ram_delta}

# XGBOOST TRAINING
XGB_PARAMS = {
    "n_estimators": 2000, "learning_rate": 0.03, "max_depth": 8,
    "reg_lambda": 3,
    "objective": "binary:logistic", "eval_metric": ["logloss", "auc"],
    "random_state": 42,
    "early_stopping_rounds": 100, "scale_pos_weight": 1.5,
    "enable_categorical": True, "tree_method": "hist", "device": "cuda",
    "max_bin": 128,
}

def train_xgboost():
    MODELS_DIR.mkdir(exist_ok=True); REPORTS_DIR.mkdir(exist_ok=True)
    df, feats, cats = load_and_prep_data(to_category=True)

    X_draft, y_draft, _, _ = get_holdout_split(df.sample(frac=0.2,
        random_state=42), feats)
    draft = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1,
        max_depth=6,
                                random_state=42, tree_method="hist",
        enable_categorical=True, n_jobs=-1)
    draft.fit(X_draft, y_draft, verbose=False)
    imp = pd.DataFrame({"Feature": feats, "Score":
        draft.feature_importances_})
    good = imp[imp["Score"] > 0]["Feature"].tolist()
    cats = [c for c in cats if c in good]
    df.drop(columns=[c for c in feats if c not in good], inplace=True)
    del X_draft, y_draft, draft; gc.collect()

    X_train, y_train, X_test, y_test = get_holdout_split(df, good)
    del df; gc.collect()
    model = xgb.XGBClassifier(**XGB_PARAMS)
    perf = _measure_fit(model, lambda m: m.fit(X_train, y_train,
                                                eval_set=[(X_train, y_train),
        (X_test, y_test)], verbose=100))
    psi = calculate_psi(model.predict_proba(X_train)[: , 1],
        model.predict_proba(X_test)[: , 1])
    print(f"PSI: {psi:.4f}")
    del X_train, y_train; gc.collect()

    evaluate_and_save(model, X_test, y_test, good, perf, "xgboost")

    er = model.evals_result()
    if "validation_0" in er and "validation_1" in er:
        plt.figure(figsize=(10, 6))
        plt.plot(er["validation_0"]["logloss"], label="Train", color="blue")
        plt.plot(er["validation_1"]["logloss"], label="Validation",
        color="orange")
        plt.title("Loss Curve - XGBOOST"); plt.xlabel("Iterations");
    plt.ylabel("Logloss")
    plt.legend(); plt.grid(alpha=0.4); plt.tight_layout()
    plt.savefig(REPORTS_DIR / f"{TIMESTAMP}_xgboost_loss_curve.png",
        facecolor="white"); plt.close()

```

```

generate_shap_plots(model, X_test, "xgboost")
model.save_model(MODELS_DIR / f"{TIMESTAMP}_xgboost_model.json")

# LIGHTGBM TRAINING
LGBM_PARAMS = {
    "n_estimators": 2000, "learning_rate": 0.03, "max_depth": 8,
    "num_leaves": 64,
    "reg_lambda": 3, "objective": "binary", "metric": ["auc",
    "binary_logloss"],
    "random_state": 42, "n_jobs": 2, "class_weight": "balanced",
    "importance_type": "gain",
}

def train_lgbm():
    MODELS_DIR.mkdir(exist_ok=True); REPORTS_DIR.mkdir(exist_ok=True)
    df, feats, cats = load_and_prep_data(to_category=True)

    X_draft, y_draft, _, _ = get_holdout_split(df, feats)
    draft = lgb.LGBMClassifier(n_estimators=100, learning_rate=0.1,
max_depth=6, num_leaves=31,
                                random_state=42, class_weight="balanced",
importance_type="gain", n_jobs=2)
    draft.fit(X_draft, y_draft, categorical_feature=cats)
    imp = pd.DataFrame({"Feature": feats, "Score":
draft.feature_importances_})
    good = imp[imp["Score"] > 0]["Feature"].tolist()
    cats = [c for c in good if c in cats]
    del X_draft, y_draft, draft; gc.collect()

    X_train, y_train, X_test, y_test = get_holdout_split(df, good)
    del df; gc.collect()
    model = lgb.LGBMClassifier(**LGBM_PARAMS)
    perf = _measure_fit(model, lambda m: m.fit(
        X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test)],
        eval_names=["Train", "Test"], categorical_feature=cats,
        callbacks=[lgb.early_stopping(stopping_rounds=100, verbose=True)]))
    psi = calculate_psi(model.predict_proba(X_train)[:, 1],
model.predict_proba(X_test)[:, 1])
    print(f"PSI: {psi:.4f}")
    del X_train, y_train; gc.collect()

    evaluate_and_save(model, X_test, y_test, good, perf, "lgbm")

    er = getattr(model, "evals_result_", getattr(model, "_evals_result", {}))
    keys = list(er.keys())
    if len(keys) >= 2:
        metric = "binary_logloss" if "binary_logloss" in er[keys[0]] else
"logloss"
        if metric in er[keys[0]]:
            plt.figure(figsize=(10, 6))
            plt.plot(er[keys[0]][metric], label="Train", color="blue")
            plt.plot(er[keys[1]][metric], label="Validation", color="orange")
            plt.title("Loss Curve - LGBM"); plt.xlabel("Iterations");
plt.ylabel("Loss")
            plt.legend(); plt.grid(alpha=0.4); plt.tight_layout()
            plt.savefig(REPORTS_DIR / f"{TIMESTAMP}_lgbm_loss_curve.png",
facecolor="white"); plt.close()

    generate_shap_plots(model, X_test, "lgbm")
    joblib.dump(model, MODELS_DIR / f"{TIMESTAMP}_lgbm_model.pkl")

```

```

# CATBOOST TRAINING
CATBOOST_PARAMS = {
    "iterations": 2000, "learning_rate": 0.03, "depth": 8, "l2_leaf_reg": 3,
    "loss_function": "Logloss", "eval_metric": "AUC", "random_seed": 42,
    "task_type": "CPU", "early_stopping_rounds": 100, "verbose": 100,
    "auto_class_weights": "Balanced", "allow_writing_files": False,
}

def train_catboost():
    MODELS_DIR.mkdir(exist_ok=True); REPORTS_DIR.mkdir(exist_ok=True)
    df, feats, cats = load_and_prep_data(to_category=False)

    X_draft, y_draft, _, _ = get_holdout_split(df, feats)
    draft = CatBoostClassifier(iterations=100, learning_rate=0.1, depth=6,
                               random_seed=42,
                               verbose=False, allow_writing_files=False,
                               auto_class_weights="Balanced")
    draft_pool = Pool(X_draft, y_draft, cat_features=cats)
    draft.fit(draft_pool)
    imp = pd.DataFrame({"Feature": feats,
                        "Score": draft.get_feature_importance(draft_pool,
type="PredictionValuesChange")})
    good = imp[imp["Score"] > 0]["Feature"].tolist()
    cats = [c for c in good if c in cats]
    del X_draft, y_draft, draft, draft_pool; gc.collect()

    X_train, y_train, X_test, y_test = get_holdout_split(df, good)
    del df; gc.collect()
    train_pool = Pool(X_train, y_train, cat_features=cats)
    test_pool = Pool(X_test, y_test, cat_features=cats)
    model = CatBoostClassifier(**CATBOOST_PARAMS)
    perf = _measure_fit(model, lambda m: m.fit(train_pool,
eval_set=test_pool, use_best_model=True, plot=False))
    psi = calculate_psi(model.predict_proba(train_pool)[: , 1],
model.predict_proba(test_pool)[: , 1])
    print(f"PSI: {psi:.4f}")
    del X_train, y_train, train_pool; gc.collect()

    evaluate_and_save(model, X_test, y_test, good, perf, "catboost")

    er = model.get_evals_result()
    if "learn" in er and "validation" in er:
        plt.figure(figsize=(10, 6))
        plt.plot(er["learn"]["Logloss"], label="Train", color="blue")
        plt.plot(er["validation"]["Logloss"], label="Validation",
color="orange")
        plt.title("Loss Curve - CATBOOST"); plt.xlabel("Iterations");
plt.ylabel("Logloss")
        plt.legend(); plt.grid(alpha=0.4); plt.tight_layout()
        plt.savefig(REPORTS_DIR / f"{TIMESTAMP}_catboost_loss_curve.png",
facecolor="white"); plt.close()

    generate_shap_plots(model, X_test, "catboost")
    model.save_model(str(MODELS_DIR / f"{TIMESTAMP}_catboost_model.cbm"))

# ENTRY POINTS
if __name__ == "__main__":
    import argparse

```

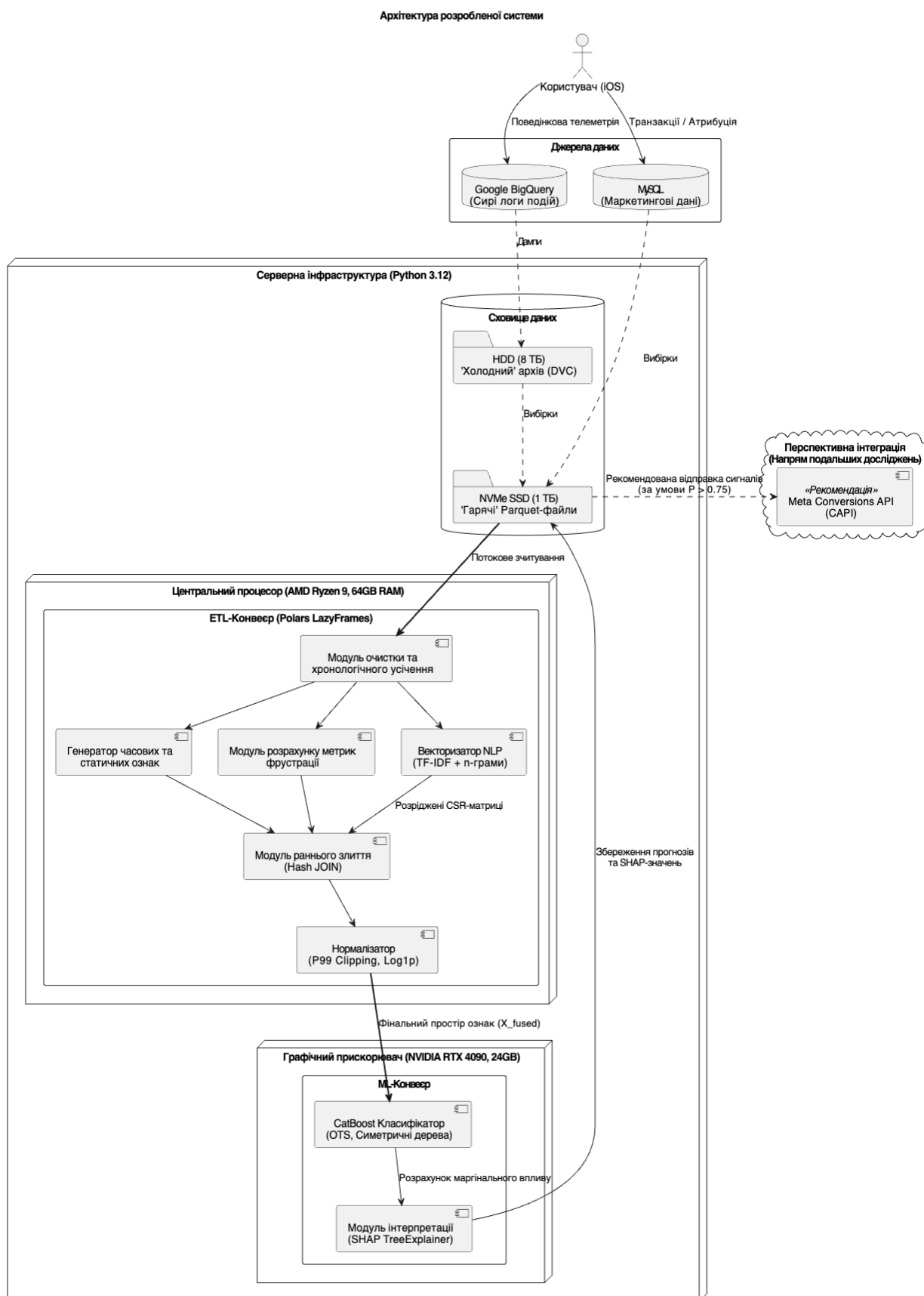
```

parser = argparse.ArgumentParser()
parser.add_argument("task", choices=["etl", "stats", "build", "xgboost",
"lgbm", "catboost"])
parser.add_argument("--source", default="file", choices=["file", "db"])
parser.add_argument("--force", action="store_true")
parser.add_argument("--date", default=None)
parser.add_argument("--only", default=None)
args = parser.parse_args()

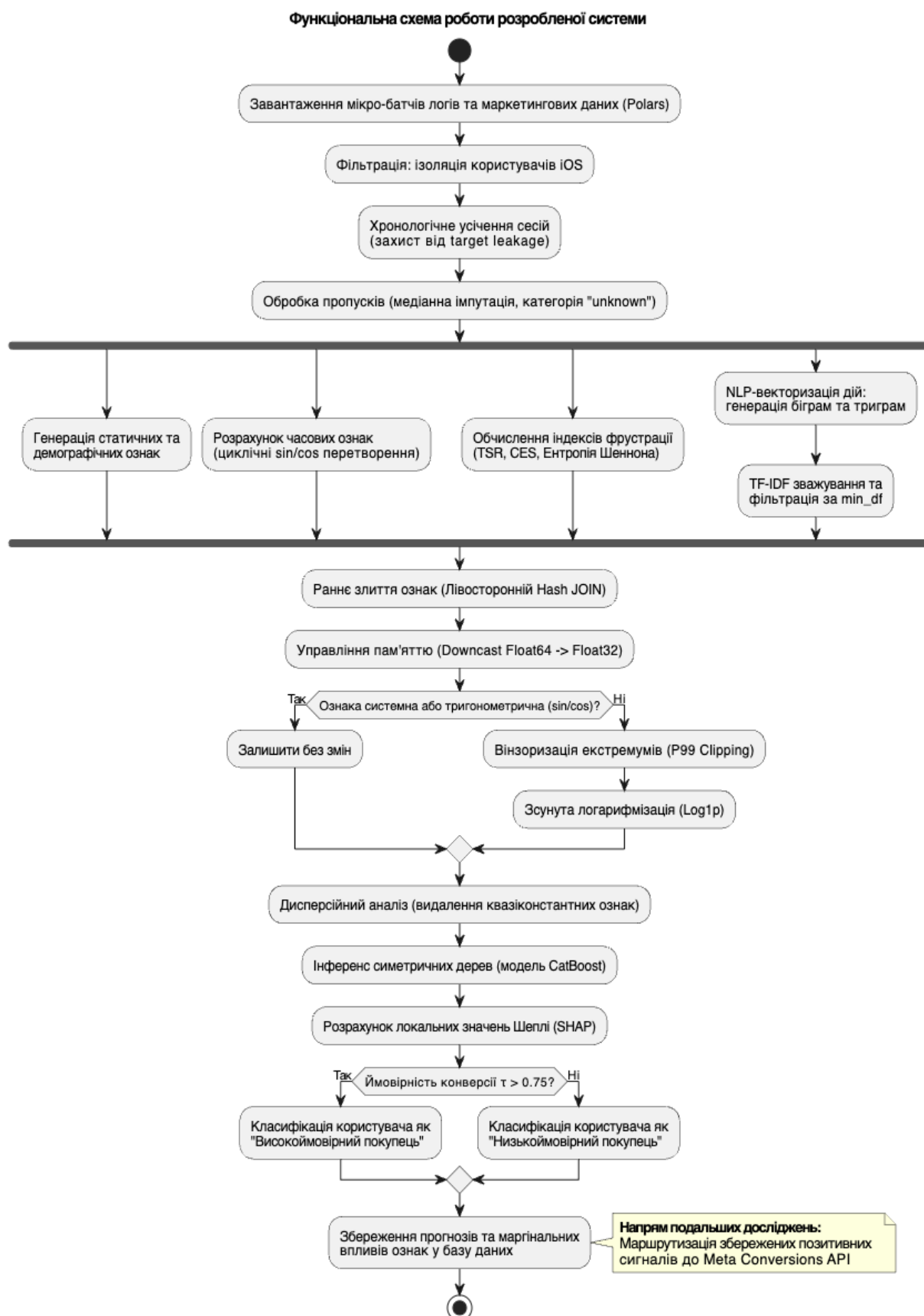
target_dt = (datetime.strptime(args.date, "%Y-%m-%d").date()
              if args.date else datetime.now().date() - timedelta(days=1))

if args.task == "etl":
    HistoricalDataLoader().run_pipeline(
        start_date=datetime(2023, 6, 1),
        final_end_date=datetime(2025, 11, 15),
        split_date=datetime(2025, 4, 20),
    )
elif args.task == "stats":
    run_stats_update(target_dt, source_mode=args.source,
                    force_restart=args.force,
specific_transformer=args.only)
elif args.task == "build":
    build_main()
elif args.task == "xgboost":
    train_xgboost()
elif args.task == "lgbm":
    train_lgbm()
elif args.task == "catboost":
    train_catboost()

```



ДОДАТОК В ФУНКЦІОНАЛЬНА СХЕМА РОБОТИ СИСТЕМИ



ДОДАТОК Г ПРЕЗЕНТАЦІЯ

**МОДЕЛІ ПОВЕДІНКОВОЇ АНАЛІТИКИ І МАШИННОГО
НАВЧАННЯ ДЛЯ КОНСТРУЮВАННЯ ОЗНАК І
ПРОГНОЗУВАННЯ КОНВЕРСІЇ КОРИСТУВАЧІВ
МОБІЛЬНОГО ЗАСТОСУНКУ**

Виконала:
студентка групи КА-22
Гаврилюк Б.О.

Керівник:
професор кафедри ММСА, д.т.н., проф.
Недашківська Н. І.

Об'єкт дослідження

Процес конверсії користувачів утилітарних мобільних застосунків та їхні складні поведінкові патерни в межах перших сесій.

Предмет дослідження

Методи Feature Engineering, алгоритми обробки великих даних (Polars) та ансамблеві моделі (XGBoost, LightGBM, CatBoost) для задач бінарної класифікації.

Мета роботи

Підвищення ефективності залучення та монетизації користувачів шляхом розробки системи раннього прогнозування конверсії для динамічної оптимізації рекламних бюджетів через Conversions API (CAPI).

АКТУАЛЬНІСТЬ РОБОТИ

Прогнозування ймовірності конверсії користувачів у перші години використання додатка дозволяє бізнесу оперативно оцінювати якість залученого трафіку без очікування завершення тестового періоду. Це зміщує фокус аналітики з констатації факту покупки на проактивне прогнозування результату.

Інтеграція прогнозної моделі з даними маркетингових каналів забезпечує динамічну оптимізацію рекламних бюджетів. Замість оцінки кампаній за вартістю встановлення, компанія отримує змогу оптимізувати закупівлю трафіку на основі прогнозованої окупності та ймовірності конверсії конкретних когорт, джерел чи креативів. Це дозволяє в режимі реального часу масштабувати прибуткові рекламні кампанії та відключати ті, що генерують нецільовий трафік.

3

ПОСТАНОВКА ЗАДАЧІ

- 1) Розробка ETL-конвеєра на базі Polars для інтеграції логів без перевантаження RAM;
- 2) Реалізація партиціювання та батчингу за когортами для економії ресурсів процесора;
- 3) Семантичне тегування подій на основі багаторівневої таксономії;
- 4) Перехід від абсолютних лічильників до відносних;
- 5) Векторизація поведінкових послідовностей за допомогою TF-IDF;
- 6) Агрегація когортних статистик для нормалізації індивідуальної поведінки користувачів;
- 7) Навчання моделі градієнтного бустингу на згенерованому просторі ознак;
- 8) Валідація якості моделі бінарної класифікації;
- 9) Глобальна та локальна інтерпретація результатів за допомогою SHAP;
- 10) Калібрування асиметричного порогу.

4

ФОРМАЛЬНА ПОСТАНОВКА ЗАДАЧІ

- Цільовою функцією системи є знаходження відображення $f(X) \rightarrow P(y = 1|X)$, де P — апостеріорна умовна ймовірність виконання користувачем цільової дії.
- Цільова змінна — $y \in \{0,1\}$, де $y = 1$ позначає факт успішної оплати підписки після завершення безкоштовного тестового періоду (міноритарний клас); $y = 0$ — відсутність оплати (мажоритарний клас).
- Багатовимірний простір ознак описується вектором $X = [X_{static}, X_{time}, X_{context}, X_{tfidf}]$, де X_{static} — категоріальні демографічні та маркетингові атрибути; X_{time} — тригонометричні темпоральні координати розподілу активності; $X_{context}$ — індекси взаємодії з продуктом; X_{tfidf} — високорозмірний розріджений вектор хронологічних послідовностей дій користувача.
- Обмеженням є фіксоване вікно спостереження після початку тестового періоду Δt , яке є меншим за тривалість тестового періоду, для захисту від витоку даних.

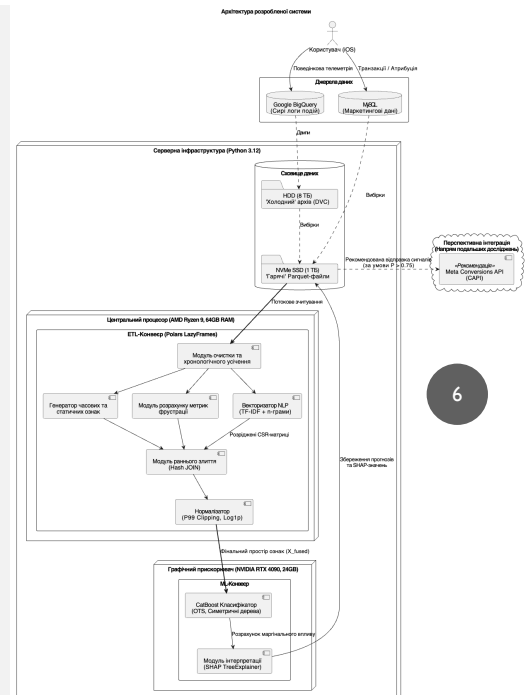
5

АРХІТЕКТУРА РОЗРОБЛЕНОЇ СИСТЕМИ

Обчислювальне ядро системи проектувалося з урахуванням високих навантажень під час обробки великих масивів поведінкових логів. Для паралелізації потокових аналітичних операцій, агрегації та ресурсомістких хеш-з'єднань у Polars використовується 16-ядерний процесор AMD Ryzen 9 5950X із 32 потоками інструкцій та уніфікованою кеш-пам'яттю L3 обсягом 64 МБ.

Стадія безпосереднього навчання та налаштування ансамблів градієнтного бустингу повністю перекладається на графічний прискорювач NVIDIA GeForce RTX 4090. Наявність 24 ГБ надшвидкої відеопам'яті VRAM та понад 16 тисяч ядер CUDA дозволяє розпаралелювати математичні обчислення й безперервно утримувати тренувальні датасети у відеопам'яті.

Для усунення вузьких місць на етапі вводу-виводу та захисту системи від браку оперативної пам'яті (RAM 64 ГБ) реалізовано дворівневу архітектуру зберігання даних. "Гарячий" рівень організовано на швидкісному накопичувачі NVMe SSD (1024 ГБ) для Parquet-файлів та проміжних обчислень, а "холодний" рівень — на архівному HDD (8 ТБ) під управлінням інструменту версіонування великих даних DVC.



6

ВИБІР ОБЧИСЛЮВАЛЬНОГО ЯДРА

Перехід від традиційної бібліотеки Pandas до архітектури Polars і колонкового формату Apache Arrow обрано як необхідну умову для забезпечення стійкості системи до браку оперативної пам'яті, прискорення ETL-операцій та мінімізації обчислювальних витрат за допомогою лінійних обчислень і нативної багатопотоковості. Порівняльний аналіз архітектурних переваг для обробки надвеликих масивів логів:

Характеристика	Pandas (Стандарт)	Polars (Розроблено)
Модель виконання	Жадібна (негайна матеріалізація)	Лінива (оптимізація графа DAG)
Паралелізм	Однопотоковий (GIL limit)	Нативна багатопотоковість (Rust)
Формат пам'яті	NumPy (рядковий)	Apache Arrow (колонковий)
Використання RAM	Високе (в 5-10 разів більше даних)	Оптимізоване (потоків агрегація)
Оптимізація запитів	Відсутня	Predicate & Projection Pushdown

9

ПОРІВНЯЛЬНИЙ МАТЕМАТИЧНИЙ ТА
ОБЧИСЛЮВАЛЬНИЙ АНАЛІЗ АРХІТЕКТУР
ПАМ'ЯТІ

Колонкова архітектура Apache Arrow, на відміну від рядкової, мінімізує витрати введення-виведення за рахунок зчитування лише цільових стовпців і забезпечує високу локальність кешу для векторизації обчислень за апаратним принципом SIMD.

Характеристика	Рядкова архітектура	Колонкова архітектура
Фізичне розміщення	Суміжне розміщення атрибутів одного запису	Суміжне розміщення одного атрибута для всіх записів
Вартість запити	$C_{row} = N \cdot \sum_{j=1}^M s_j$	$C_{col} = N \cdot \sum_{j \in K} s_j$
Локальність кешу	Низька, забруднення кешу непотрібними атрибутами	Висока, кешуються лише цільові дані
Оптимізація процесора	SISD (Single Instruction, Single Data)	SIMD (Single Instruction, Multiple Data)
Рівень стиснення	Низький, гетерогенні типи даних у блоці	Високий, гомогенні масиви значень

10

ФУНКЦІОНАЛЬНА СХЕМА СИСТЕМИ

Конвеєр обробки даних починається з фільтрації сирих логів, дедуплікації подій та виділення цільових користувачів. Для запобігання витку цільової змінної застосовується метод хронологічного усічення подій до моменту фіксації цільової дії.

На основі очищених даних формуються незмінні демографічні атрибути та маркетингові характеристики пристрою. Часові параметри сесій проходять тригонометричну трансформацію (обчислення синусу й косинусу години), що дозволяє моделі коректно вловлювати циклічні добові патерни активності.

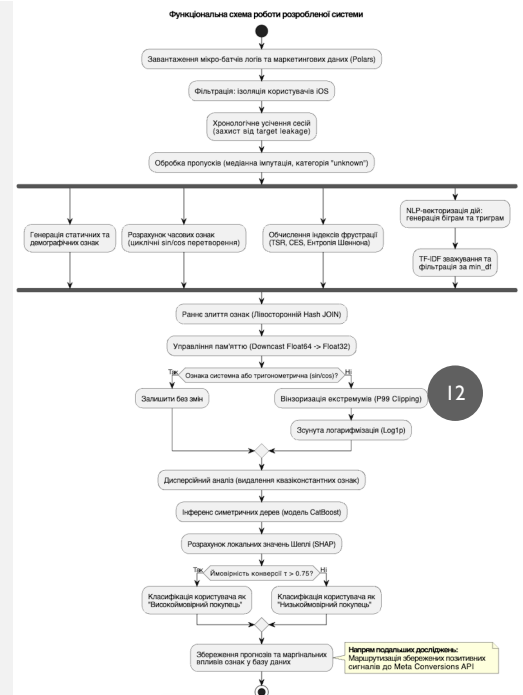
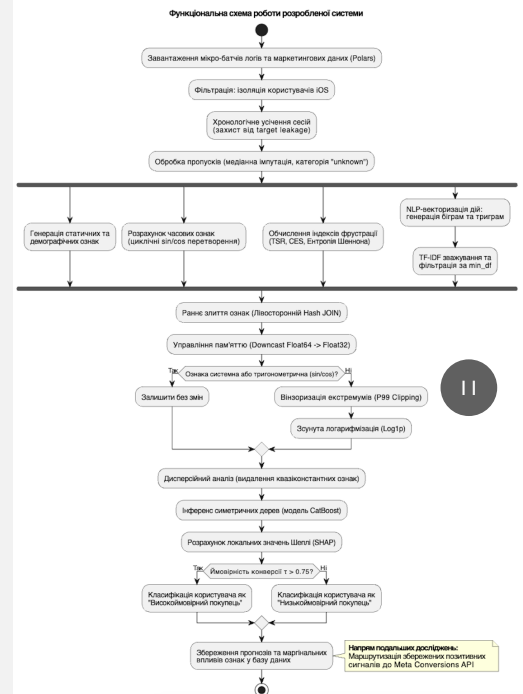
Здійснюється семантичне тегування сирих назв подій за допомогою багаторівневої таксономії подій інтерфейсу. До ознакового простору додаються індекси успішності сценаріїв (TSR), коефіцієнти когнітивного зусилля (CES) та ентропія Шеннона для кількісної оцінки хаотичності й циклічності рухів користувача.

ФУНКЦІОНАЛЬНА СХЕМА СИСТЕМИ

Для збереження інформації про хронологічний порядок взаємодій послідовності логів трансформуються за аналогією до тексту, де події виступають у ролі слів. Алгоритм генерує біграми і триграми подій, які проходять статистичне зважування за метрикою TF-IDF для виділення унікальних сигналів.

Усі згенеровані модальності об'єднуються у межах єдиної матриці за парадигмою раннього злиття на рівні ознак (Feature-Level Fusion). Для оптимізації обчислень числові блоки примусово приводяться до формату Float32, проходять вінзоризацію (P99 Clipping) та зсунуту логарифмізацію (Log1p) для стабілізації розподілів з важкими хвостами.

Скрипт виконує дисперсійний аналіз і безповоротно видаляє квазіконстантні ознаки з нульовою інформаційною цінністю. Фінальний простір ознак подається на вхід класифікатора моделі, який за допомогою симетричних дерев обчислює ймовірність конверсії користувача.



МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ
ПОВЕДІНКОВИХ МЕТРИК ФРУСТРАЦІЇ

$$TSR = \frac{\sum_{i=1}^N \mathbb{I}(e_i \in E_{success})}{\sum_{i=1}^N \mathbb{I}(e_i \in E_{attempt})}$$

$$CES = \frac{\sum_{j=1}^S N_{total_actions_j}}{\sum_{j=1}^S N_{unique_actions_j}}$$

$$H(A) = - \sum_{i=1}^k p(a_i) \log_2 p(a_i)$$

Task Success Rate (TSR) визначає умовну ймовірність того, що ініційований користувачем функціональний сценарій завершиться успішним результатом. Обчислюється за допомогою індикаторної функції, яка фіксує входження події e_i у підмножину успіхів $E_{success}$ відносно загальної кількості спроб $E_{attempt}$ у межах аналізованої сесії. Значення локальне відхилення TSR індивіда від медіанних значень його когорти слугує потужним предиктором відмови від подальшої конверсії.

Cognitive Effort Score (CES) визначає рівень розгубленості користувача через оцінку надлишковості й циклічності його рухів в інтерфейсі. Обчислюється як математичне відношення сумарної кількості дій $N_{total_actions_j}$ до кількості унікальних категорій подій $N_{unique_actions_j}$ у межах сесії S . Для впевненої, лінійної поведінки $CES \rightarrow 1$. Значення перевищення одиниці є математичним доказом генерування надлишкових дій, спричинених нерозумінням логіки екранів або технічними збоями.

Ентропія дій Шеннона $H(A)$ — це інформаційно-теоретична оцінка хаотичності рухів, де $p(a_i)$ — емпірична відносна частота виникнення події i -ї категорії ієрархічної таксономії в сесії. Низька ентропія свідчить про монотонну концентрацію, тоді як висока нормалізована ентропія є беззаперечним симптомом розпорошеності уваги та дезорієнтації користувача під час онбордингу.

КОНСТРУЮВАННЯ ПОВЕДІНКОВИХ ОЗНАК
ТА МЕТРИК ФРУСТРАЦІЇ

Замість використання простих лічильників дій, система трансформує дискретні події у неперервні математичні показники:

- Статичні ознаки: демографічні та маркетингові атрибути із генерацією перехресних взаємодій та групуванням рідкісних категорій для зниження кардинальності.
- Темпоральні ознаки: циклічні тригонометричні перетворення часу ($\sin/\cos$), розрахунок тривалості сесій, затримок між діями та відносних відхилень від когортних значень.
- Метрики фрустрації: індекс успішності завдань, оцінка когнітивного зусилля та ентропія дій за Шенноном для ідентифікації хаотичної поведінки.
- Векторизація NLP: адаптація методу TF-IDF для хронологічних послідовностей, що дозволяє зважувати локальну значимість подій та фільтрувати глобальний статистичний шум.

Категорія ознак	Опис та семантичне значення	Алгоритм обчислення та логіка скрипта
Статичні	Незмінні атрибути пристрою та інсталяції.	Вилучаються з метаданих першого логу за <code>user_id</code> . Рідкісні значення групуються в «Other» за заданим.
Часові	Розподіл активності в часі, частота сесій та загальна тривалість залученості.	Обчислюються віконними функціями як різниця між сусідніми часовими мітками.
Поведінкові	Кількісні метрики інтенсивності взаємодії до моменту усічення.	Групування за <code>user_id</code> та <code>session_id</code> . Підрахунок частот подій, середніх і максимальних значень.
Контекстні	Фіксують взаємодію зі специфічними вузлами застосунку.	Фільтрація подій за таксономією. Генерація бінарних прапорців відвідування та кумулятивних сум взаємодій з модулями.

СТАТИСТИЧНІ ЗАСАДИ ВЕКТОРИЗАЦІЇ ТА
СТАБІЛІЗАЦІЇ ОЗНАК

Для збереження хронологічної послідовності та транзакційного контексту переходів застосовуються інструменти обробки природної мови (NLP). Хронологічний лог подій користувача трансформується за ієрархічною таксономією у текстовідібний "документ" d , а окремі івенти або послідовності — у "терміни" t у межах загального корпусу когорти D . Статистична адаптація TF-IDF обчислює локальну частоту терміна $TF(t,d)$, нормалізовану для усунення зміщення на користь аномально довгих сесій, та обернену частоту документа $IDF(t,D)$ з логарифмічним згладжуванням знаменника для унеможливлення ділення на нуль. Підсумкова вага $TFIDF(t,d,D)$ логарифмічно штрафує тривіальні масові події, які зустрічаються у більшості профілів, і максимізує вагу унікальних, рідкісних предиктивних сигналів.

$$TF(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}; \quad IDF(t,D) = \log \frac{|D|}{|\{d \in D: t \in d\}| + 1}; \quad TFIDF(t,d,D) = TF(t,d) \cdot IDF(t,D)$$

15

Динамічні часові показники поведінки характеризуються асиметричними розподілами з важкими хвостами через наявність аномальних користувачів. Для нейтралізації цього явища реалізовано вінзоризацію (P99 Clipping) — примусове обмеження значень 99-м процентилем популяції, що захищає градієнти від вибухів під час оптимізації, та зсунуту логарифмічну трансформацію LogIp, яка лінеаризує експоненційні залежності й унеможливорює помилку для нульових значень розріджених матриць. Перетворення відображає закон спадної граничної корисності, допомагаючи гістограмним оптимізаторам бустингу рівномірно розбивати ознаки на кошики.

$$x_i^* = \min(x_i, P_{0.99}(X)) \qquad x'' = \ln(1 + x_i^*)$$

ВПЛИВ СТРАТЕГІЙ
ПРОСТОРОВОЇ
НОРМАЛІЗАЦІЇ НА
ГЕТЕРОГЕННІ ТИПИ
ОЗНАК

Застосовані стратегії просторової нормалізації забезпечують диференційовану стабілізацію дисперсії простору ознак залежно від їхньої семантичної природи. Системні ідентифікатори та тригонометричні темпоральні координати повністю ізольовані від Clipping та LogIp-перетворень для збереження цілісності ключів злиття та кругової часової геометрії. Для числових поведінкових показників та метрик фрустрації вінзоризація за 99-м процентилем та зсунуте логарифмування компресують важкі хвости розподілу, що оптимізує побудову гістограм алгоритмами бустингу та унеможливорює перенавчання на аномальних викидах. Текстові TF-IDF вектори зазнають делікатного згладжування домінантних слів із гарантованим збереженням розрідженості матричної структури.

Категорія вхідних ознак	Вплив P99 Clipping	Вплив LogIp трансформації	Загальний ефект на простір ознак
Системні ідентифікатори	Ігнорується алгоритмом	Ігнорується алгоритмом	Збереження цілісності ключів злиття.
Темпоральні	Захищено від застосування (фільтр sin/cos)	Захищено від застосування	Збереження тригонометричної геометрії [-1, 1].
Абсолютні поведінкові	Відсікання аномальних ботів/користувачів-китів	Компресія важкого "хвоста" розподілу	Покращення побудови гістограм, уникнення перенавчання.
Текстові TF-IDF Вектори	Незначний (значення зазвичай < 1)	Незначний (компресія дробових значень)	Згладжування домінуючих слів, збереження розрідженості.
Відносні метрики	Обмеження до 100% або верхньої межі когорти	Переведення лінійної шкали в логарифмічну	Посилення градієнта для початкових значень успішності.

16

МАКРОСТРУКТУРА, ТОПОЛОГІЯ ТА ТРАНСФОРМАЦІЙНІ СТАНИ ФІНАЛЬНОГО ДАТАСЕТУ ПІСЛЯ ІНТЕГРАЦІЇ

Фінальний інтегрований простір ознак об'єднує статичні маркетингові атрибути, тригонометричні темпоральні координати, агреговані індикатори залученості, індекси фрустрації та розріджені TF-IDF вектори послідовностей дій користувача. Логіка обробки кожної категорії ознак передбачає диференційовану оптимізацію: числові та контекстні метрики стабілізовано через вінзорування (P99 Clipping) і зсунуту логарифмізацію (Log1p) у форматі Float32, темпоральні координати збережено у вихідній круговій геометрії [-1, 1], а категоріальні змінні та розріджені NLP-структури пройшли процедури зниження кардинальності, імпутації пропусків та дисперсійного відсікання квазіконстантних ознак для захисту моделі від перенавчання.

Кластер ознак у датасеті	Приклади змінних у просторі Xfused	Алгоритмічні трансформації та фінальний стан у матриці
Системно-ідентифікаційні ключі	user_id, original_purchase_date, is_converted	Залишені без змін. Захищені масками виключення. Слугують опорними ключами для DVC та таргет-лейблами для розрахунку градієнтів.
Статичні (Демографія та Маркетинг)	country, media_source, device_type, inter_country_device	Приведено до типу Categorical. Рідкісні екземпляри алгоритмічно об'єднані у групу "Other" для зниження кардинальності. Заповнення null-значень статичною заглушкою "unknown".
Тимчасово-циклічні координати (Temporal)	sin_hour_install, cos_day_of_week	Float32. Виключені з процесів P99 Clipping та Log1p для збереження тригонометричної кругової геометрії [-1, 1].
Агреговані поведінкові індикатори (Behavioral)	total_events_count, avg_session_duration, ratio_spend_to_cohort	Float32. Медіанна імпутація пропусків. Усічення екстремумів за 99-м процентилем та зсунута логарифмізація.
Метрики фрустрації та когнітивного навантаження (Contextual)	user_tsr (Task Success Rate), ces_score (Cognitive Effort), action_entropy	Float32. Стабілізація дисперсії через логарифмічне згладжування. Нормалізація формули Шеннона (0-1) для ентропії шляхів користувача.
Обробка природної мови (Синтезовані TF-IDF вектори)	tfidf_view_paywall, tfidf_error_click_retry	Розріджені компоненти перетворені у Float32. Пройшли попередню min_df фільтрацію на етапі екстракції NLP та фінальне квазіконстантне відсікання.

17

ХАРАКТЕРИСТИКИ GBDT МОДЕЛЕЙ

• **XGBoost**

Використовує апроксимацію Тейлора другого порядку для оптимізації функції втрат. Найкраще калібрування ймовірностей, але найповільніше навчання та високе споживання RAM через DMatrix структури.

• **LightGBM**

Використовує стратегію росту дерева по листках (leaf-wise). Найвища повнота виявлення та швидкість навчання завдяки GOSS та EFB алгоритмам. Схильний до перенавчання на малих вибірках.

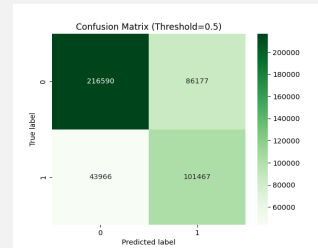
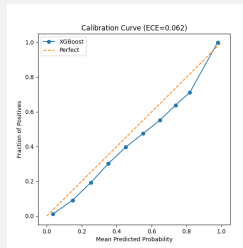
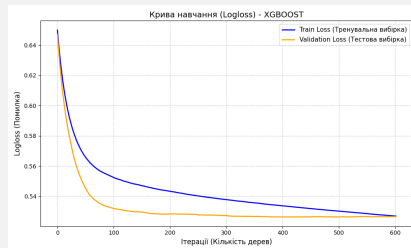
• **CatBoost**

Базується на симетричних деревах (Oblivious Trees). Нативна обробка категоріальних даних через Ordered Target Statistics (OTS). Найшвидший інференс та висока стійкість до дрейфу даних.

Алгоритм	Основна математична оптимізація	Механізм боротьби з розрідженістю	Стратегія розвитку дерева
XGBoost	Апроксимація Тейлора 2-го порядку	Sparsity-aware split finding	Зростання по рівнях
LightGBM	Семплінг за градієнтом GOSS	Злиття графів ознак EFB	Зростання по листках
CatBoost	Обчислення цільових статистик OTS через перестановки σ	Комбінаторика та злиття категорій	Симетрична топологія

18

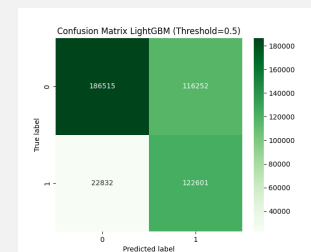
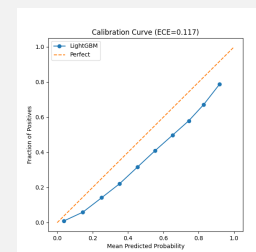
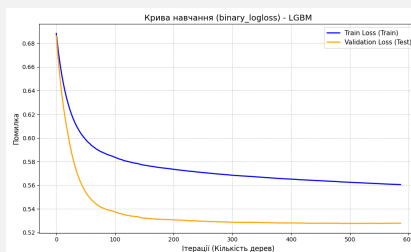
НАВЧАННЯ МОДЕЛІ XGBOOST



Навчання моделі XGBoost характеризується високою стабільністю та плавним асимптотичним зниженням функції втрат без ознак перенавчання. Завдяки симетричному рівневому росту дерев модель демонструє найкраще ймовірнісне калібрування, де крива прогнозів майже ідеально накладається на еталонну діагональ. Проте на матриці помилок помітний консервативний характер класифікації: модель забезпечує найвищу точність, але суттєво втрачає у повноті.

19

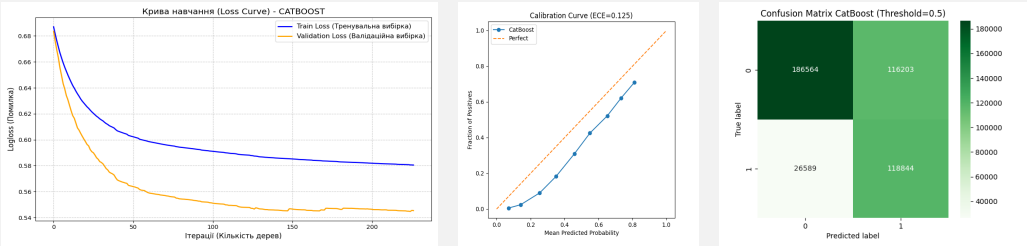
НАВЧАННЯ МОДЕЛІ LIGHTGBM



Крива навчання LightGBM фіксує стрімке експоненційне падіння похибки на перших 100 ітераціях з подальшим виходом на стабільне плато. Матриця помилок підтверджує максимальну предиктивну спроможність моделі у виявленні міноритарного класу, забезпечуючи найвищу повноту та піковий показник MCC. Однак через жадібну leaf-wise топологію калібрувальна крива виявляє суттєвий ефект гіперпевненості, коли модель систематично завищує ймовірність відносно реальної частоти конверсій.

20

НАВЧАННЯ МОДЕЛІ CATBOOST



21

Процес мінімізації похибки CatBoost демонструє прискорену збіжність уже після 200 ітерацій, де невеликі коливання кривої відображають роботу механізму Ordered Boosting, що діє як додатковий регуляризатор проти перенавчання. Матриця помилок фіксує найбільш збалансований і комерційно виправданий компроміс між точністю та високою повнотою. При цьому симетрична геометрія дерев дозволила утримати задовільний рівень калібрування ймовірностей уздовж робочої діагоналі.

ПОРІВНЯЛЬНИЙ АНАЛІЗ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ

Оцінювання проводилося на ідентичній відкладеній вибірці за умов хронологічного ізолювання, що імітує реальні умови експлуатації та запобігає витоку даних. Хоча LightGBM показав найвищі результати за метриками F1-Score та MCC на етапі статичної крос-валідації, для промислової мікросервісної експлуатації було обрано архітектуру CatBoost. Завдяки механізму Ordered Target Statistics, алгоритм нативно обробляє висококардинальні категоріальні ознаки «на льоту», що позбавляє необхідності створювати ресурсомісткі ONE-пайплайни та запобігає фрагментації простору рішень. Крім того, топологія симетричних дерев забезпечує високу стійкість до перенавчання і безпрецедентний час інференсу — 5.76 мілісекунд на 1000 записів, що майже в 10 разів швидше за LightGBM.

Метрика	XGBoost	LightGBM	CatBoost
Precision	0.5407	0.5132	0.5056
Recall	0.6976	0.843	0.8171
F1-Score	0.6092	0.638	0.6247
ROC-AUC	0.7902	0.7918	0.7791
PR-AUC	0.5767	0.5775	0.5612
MCC	0.3919	0.4307	0.4062
Brier Score	0.1732	0.1846	0.1889
ECE	0.0615	0.1174	0.1249
Accuracy	0.7096	0.6896	0.6814
Log-Loss	0.5007	0.5278	0.5476
Час навчання (секунди)	2169.93	412.16	137.36
Час інференсу на 1000 записів (мс)	30.65	56.21	5.76
Зміна споживання транзитної RAM (МБ)	-174.21	142.83	1009.51

22

МАТЕМАТИЧНІ МЕТОДИ ІНТЕРПРЕТАЦІЇ ТА КАЛІБРУВАННЯ РЕЗУЛЬТАТІВ

Теорія ігрових значень Шеплі (SHAP) використовується для розрахунку справедливого маргінального внеску кожної ознаки користувача ϕ_j у відхилення фінальної предикції $f_x(S \cup \{j\})$ від базового математичного очікування популяції $f_x(S)$ по всіх можливих комбінаціях підмножин (коаліцій) ознак S . Метод єдиний строго задовольняє аксіоми локальної точності, симетрії, фіктивності та адитивності, що дозволяє деконструювати абстрактну ймовірність до рівня чистих поведінкових тригерів фрустрації чи успіху.

$$\phi_j(f, x) = \sum_{S \subseteq F \setminus \{j\}} \frac{|S|! (|F| - |S| - 1)!}{|F|!} (f_x(S \cup \{j\}) - f_x(S))$$

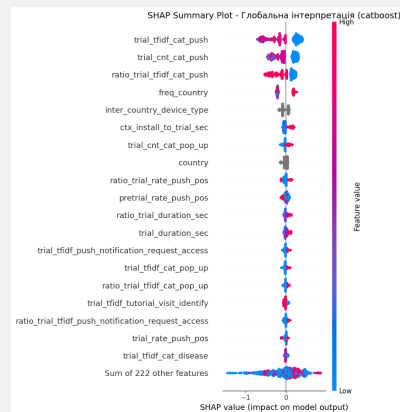
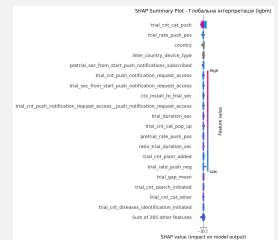
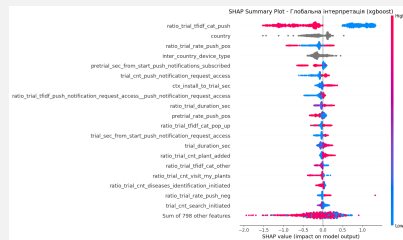
23

Для моделей, що генерують відкалібровані апостеріорні ймовірності, використано теорему, згідно з якою оптимальний поріг τ_{opt} , який максимізує гармонійне середнє (метрику F_1 -score), дорівнює половині від максимально досяжного значення F_1 на валідаційній вибірці. Це ліквідує потребу в ресурсомісткому переборі сітки порогів під час інференсу у виробничому середовищі.

$$\tau_{opt} = \frac{\max(F_1)}{2}$$

ІНТЕРПРЕТАЦІЯ ПРЕДИКТОРІВ (SHAP)

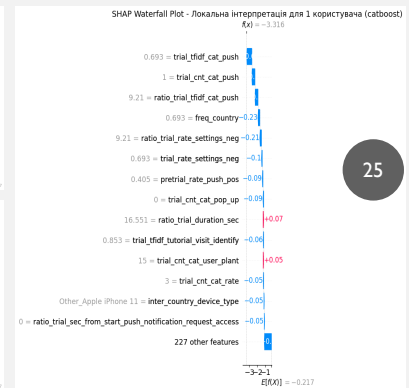
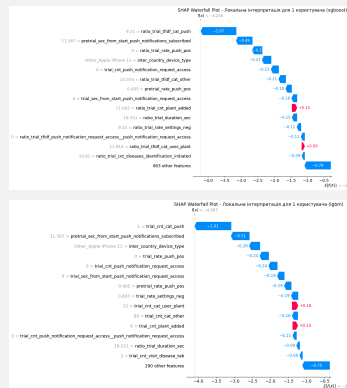
Глобальний аналіз ігрових значень Шеплі (SHAP) для моделей XGBoost, LightGBM та CatBoost демонструє високу узгодженість у визначенні ключових поведінкових чинників конверсії. Головним деструктивним фактором відтоку виступає надмірна пуш-комунікація та спам, які створюють значне когнітивне перевантаження користувача. Системні помилки та зловикористання з наданням базових дозволів також чинять негативний вплив на монетизацію. Натомість, основним чинником конверсії є глибока органічна залученість до ключого функціоналу додатку, що підтверджує важливість швидкого досягнення моменту усвідомлення цінності. Статичні демографічні та апаратні параметри задають базовий рівень платоспроможності, який коригується поточними продуктовими індексами успішності взаємодії.



24

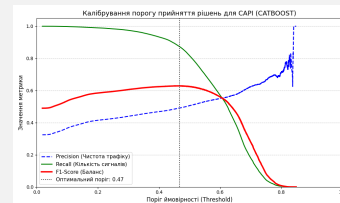
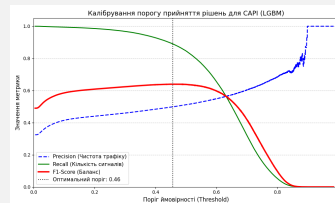
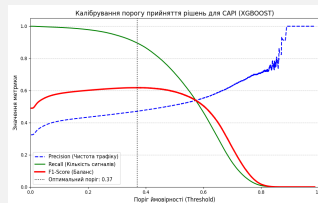
ЛОКАЛЬНА ІНТЕРПРЕТАЦІЯ МОДЕЛЕЙ

Локальна інтерпретація за допомогою каскадних діаграм SHAP фіксує процес адитивного накопичення поведінкових доказів для користувача з аномально низькою ймовірністю конверсії. Усі три моделі демонструють ідентичну топологію прийняття рішень, послідовно знижуючи прогноз від базового рівня когорти до критичного відтоку. Головним деструктивним чинником у кожній архітектурі виступає надмірний обсяг пуш-комунікації, який генерує найбільші негативні значення SHAP. Цей негативний ефект посилюється зволіканням із наданням системних дозволів на сповіщення та апаратно-демографічним контекстом пристрою. Органічна залученість користувача через взаємодію з додатком формує позитивний предиктивний імпульс, проте він виявляється математично недостатнім для компенсації фрустрації від інтерфейсу. Аналіз підтверджує здатність ансамблів GBDT чітко ізолювати конкретні "вузькі місця" та причинно-наслідкові ланцюги негативного користувацького досвіду.



25

КАЛІБРУВАННЯ ПОРОГУ ПРИЙНЯТТЯ РІШЕННЯ



26

Оптимальні математичні порогови максимізації F1-score становлять 0.37 для XGBoost, 0.46 для LightGBM та 0.47 для CatBoost, проте їх безпосереднє використання в Conversions API є бізнес-недоцільним через високий рівень хибнопозитивного шуму, що дезінформує рекламні алгоритми таргету. З метою захисту аукціону від інформаційного отруєння впроваджено асиметричний операційний поріг відсічення на рівні 0.75. Таке зміщення вправо знижує повноту виявлення до 20–30%, але максимізує точність понад 85–90%, забезпечуючи відправку серверних сигналів виключно для цільового сегмента користувачів із найвищим конверсійним потенціалом.

ІМІТАЦІЯ БІЗНЕС-НАСЛІДКІВ ПРИ ВИБОРІ РІЗНИХ ПОРОГІВ ВІДСІЧЕННЯ ДЛЯ CONVERSIONS API

Аналіз графіків калібрування та кривих Precision-Recall демонструє класичний компроміс між чистою предиктивною силою та обсягом переданих подій. Оптимальний за критерієм F1-Score поріг забезпечує статистичний баланс, проте генерує значну частку хибнопозитивних тригерів, що дезінформує оптимізатори рекламних мереж і спричиняє волатильність рентабельності. Зниження порогу відсічення до 0,3 критично засмічує потік даних "шумом", змушуючи аукціон адаптуватися під нерелевантний трафік і марно витратити бюджет. Натомість, прагматичний зсув операційного порогу вправо до рівня 0,75 свідомо знижує повноту виявлення, але максимізує точність. Таке асиметричне калібрування формує високоочищений потік преміальних "синтетичних конверсій", що радикально підвищує ROAS, прискорює навчання ML-алгоритмів таргетингу рекламних платформ та повністю унеможливує інформаційне отруєння аукціону нецільовою аудиторією.

Попір	Precision	Recall	Характеристика згенерованого масиву для CAPI	Прогнозований економічний вплив на рекламний бюджет
0.3	Низька (~0.25)	Максимальна (~0.95)	Масив екстремально засмічений. Аномальна кількість хибних спрацювань.	Експоненційне зростання вартості залучення клієнта. Алгоритм Meta оптимізується під нерелевантний трафік. Бюджет витрачається даремно.
0.47	Середня (~0.60)	Середня (~0.60)	Задовільний рівень, проте ризикований. Значна частка сигналів є "шумом".	Стандартна робота аукціону, висока волатильність рентабельності. Якість недостатня для агресивного масштабування на нових ринках.
0.75	Агресивно висока (~0.85+)	Навмисно низька (~0.25)	Високоочищений, преміальний сигнал. Гарантована якість.	Радикальне підвищення ROAS. Рекламний бюджет витрачається виключно на таргетинг аудиторії з найвищим прогнозованим життєвим циклом.

27

ВИСНОВКИ

- Відмова від рядкової парадигми на користь фреймворку Polars та колонкового формату Apache Arrow дозволила розпаралелити обробку великих масивів даних на CPU, знизити використання RAM та унеможливити помилки Out-of-Memory;
- Розроблено математичний апарат для конструювання ознак з неструктурованих логів, що дозволив обчислити неперервні метрики когнітивного навантаження та векторизувати часові послідовності за допомогою адаптованого NLP-методу TF-IDF;
- За результатами Out-of-Time крос-валідації оптимальною моделлю для промислової мікросервісної експлуатації визначено CatBoost, яка забезпечила найшвидший інференс, нативну обробку категоріальних ознак через Ordered Target Statistics та стійкість до перенавчання;
- Імплементация фреймворку SHAP дозволила ліквідувати проблему «чорної скриньки», довівши, що першопричинами відтоку користувачів є надмірне когнітивне навантаження та агресивна push-комунікація;
- На основі розрахованих кривих калібрування обґрунтовано зміщення порогу прийняття рішень до асиметричного рівня 0.75, що максимізує точність для формування очищеного потоку «синтетичних конверсій»;

28

ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

- Реалізація практичного механізму автоматизованої маршрутизації сформованих персоналізованих прогнозних сигналів безпосередньо у маркетингові платформи через Meta Conversions API;
- Розгортання системи безперервного донавчання для адаптації алгоритмів до еволюції поведінкових патернів користувачів нових версій застосунку;
- Дослідження гібридних архітектур для поєднання градієнтного бустингу з глибинним навчанням для ще глибшого захоплення часових залежностей.

29

ДЯКУЮ ЗА УВАГУ!