ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система моніторингу активності співробітників IT-компаній»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Данііл ДОРОХІН
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр.ІСДМ-61
Данііл ДОРОХІН
Ім'я, ПРІЗВИЩЕ

Керівник:
Науковий ступінь вчене звання PhD Валентина ДАНИЛЬЧЕНКО
Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання
Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інформаційні системи та технології

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ

_____Каміла СТОРЧАК
«_____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дорохін Данііл В'ячеславович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Система моніторингу активності співробітників ІТ-компаній

керівник кваліфікаційної роботи Валентина Данильченко *PhD*,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: Визначити основні вимоги та завдання, які повинні бути вирішені під час виконання дипломної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Перелік ілюстративного матеріалу: *презентація*
Вивчення функціоналу вимоги до системи

5. Перелік графічного матеріалу: *презентація*
    1. Теоретична частина
    2. Апаратні складові систем
    3. Моніторинг програмного забезпечення систем

6. Дата видачі завдання «30» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Розробка технічного завдання	30.10.2025	Виконано
2	Аналіз вимог і уточнення специфікацій	30.10.2025	Виконано
3	Розробка теоретичної частини	04.11.2025	Виконано
4	Проєктування структури ПЗ, проєктування компонентів (технічний проєкт)	10.11.2025	Виконано
5	Реалізація компонентів і автономне тестування. Збірка, комплексне тестування, оцінка (робочий проєкт)	22.11.2025	Виконано
6	Висновки за результатами аналізу	30.11.2025	Виконано
7	Розробка презентації	31.11.2025	Виконано

Здобувач вищої освіти _____ Данііл Дорохін
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Валентина Данильченко
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи: 71 сторінок, 15 рисунків, 39 джерел.

Мета роботи - полягає у розробці автоматизованої системи моніторингу активності співробітників в IT-компаніях з використанням методів машинного навчання для класифікації типу діяльності та оптимізації робочого процесу.

Об'єкт дослідження – процес моніторингу активності співробітників IT-компаній та аналіз їх продуктивності.

Предмет дослідження – автоматизована система моніторингу активності на основі машинного навчання, що інтегрує аналіз даних з пристроїв вводу та переліку запущених процесів.

Короткий зміст роботи:
У роботі представлено аналіз сучасних систем моніторингу активності персоналу та особливостей праці співробітників IT-сектору. Розроблено архітектуру системи моніторингу та проведено порівняльний аналіз алгоритмів машинного навчання для класифікації типу активності. Запропоновано використання методу випадкового лісу (Random Forest) для аналізу даних з пристроїв вводу та LSTM-мережі для обробки переліків запущених процесів. Реалізовано програмний комплекс, що забезпечує точність класифікації до 90% при мінімальному споживанні обчислювальних ресурсів. Система дозволяє автоматично класифікувати активність користувача та приймати рішення щодо обмеження доступу до робочої станції у разі виявлення непродуктивної діяльності.

КЛЮЧОВІ СЛОВА: СИСТЕМА МОНІТОРИНГУ, АКТИВНІСТЬ КОРИСТУВАЧА, МАШИННЕ НАВЧАННЯ, RANDOM FOREST, LSTM, IT-КОМПАНІЇ, ПРОДУКТИВНІСТЬ, ТЕХНОСТРЕС

ABSTRACT

Text volume of the qualification work: 71 pages, 15 figures, 39 sources.

Purpose of the work is to develop an automated employee activity monitoring system for IT companies using machine learning methods for activity type classification and work process optimization.

Object of research – the process of monitoring employee activity in IT companies and analyzing their productivity.

Subject of research – an automated activity monitoring system based on machine learning, integrating analysis of input device data and running processes lists.

Summary of the work:
The work presents an analysis of modern employee activity monitoring systems and specific features of IT sector work. A monitoring system architecture was developed and a comparative analysis of machine learning algorithms for activity type classification was conducted. The use of Random Forest method for input device data analysis and LSTM networks for processing running processes lists was proposed. A software complex was implemented, achieving classification accuracy up to 90% with minimal computational resource consumption. The system automatically classifies user activity and makes decisions about restricting workstation access when unproductive activity is detected.

KEYWORDS: MONITORING SYSTEM, USER ACTIVITY, MACHINE LEARNING, RANDOM FOREST, LSTM, IT COMPANIES, PRODUCTIVITY, TECHNOSTRESS

ЗМІСТ

Вступ

Тема роботи є актуальною з причин, описаних нижче. Робота є актуальною відповідно до завдань, визначених у розпорядженні щодо стратегічного напряму цифрової трансформації у сфері охорони здоров'я . Збір, аналіз та оцінка даних про активність користувачів — IT-працівників — є надзвичайно важливими для розуміння продуктивності та цифрового добробуту на сучасному робочому місці, зважаючи на вплив технологічного стресу.

Технологічний стрес виникає у випадках, коли працівники перевантажені надмірним обсягом інформації та завдань, які необхідно вирішувати за допомогою технологій, зокрема інформаційних. Такий стрес спричиняє наступні ефекти: емоційне виснаження та професійне вигорання, конфлікти між роботою та членами сім'ї.

Стратегіями подолання технологічного стресу є: обмеження часу взаємодії з джерелом стресу, організація графіка доступу до нього, розподіл годин доступу на довші відрізки часу, наприклад, на вихідні. Розроблена автоматизована система моніторингу активності співробітників IT-компаній дозволить у майбутньому справлятися з технологічним стресом та впливати на стан працівника.

Ступінь опрацювання теми розробки автоматизованої системи моніторингу активності людей є достатньо глибокою, щоб стверджувати наявність певної теоретичної бази для подальших досліджень. Водночас вона недостатня, щоб вважати тему повністю розкритою з усіх перспектив. Наприклад, відсутня значна кількість робіт, які досліджують особливості різних соціальних груп у контексті зазначеної автоматизованої системи, з використанням різних комбінацій підходів до збору, аналізу та оцінки даних.

Мета — розробити автоматизовану інформаційну систему моніторингу активності співробітників IT-компаній.

Завдання:

1.Вивчити науковий дискурс з теми розробки автоматизованої системи моніторингу активності людей.

2.Спроєктувати автоматизовану систему моніторингу активності співробітників ІТ-компаній.

3.Реалізувати програмний код автоматизованої системи моніторингу активності співробітників ІТ-компаній.

Об’єкт дослідження — моніторинг активності людей. Предмет дослідження — автоматизована інформаційна система моніторингу активності співробітників ІТ-компаній.

Обрані методи дослідження: аналіз, синтез, порівняння, індукція, дедукція, абстрагування, формалізація, вимірювання.

Теоретична основа — наукова література у вигляді книг, статей, патентів. *Нормативна основа* — етичні норми ІТ-спільнот, закони про персональні дані, авторське право, інформаційну безпеку в межах чинної юрисдикції. *Емпірична основа* — вимірювання результатів роботи розробленої автоматизованої системи.

Наукова новизна полягає у спеціалізації автоматизованої системи на цільовій аудиторії — співробітниках ІТ-компаній — та застосуванні нової комбінації засобів збору, аналізу й оцінки даних.

Прикладна цінність полягає в тому, що дана автоматизована система передбачає своє застосування для подальшої регуляції рівня цифрового добробуту працівника, зокрема його ефективності.

1 АНАЛІЗ СУЧАСНИХ СИСТЕМ МОНІТОРИНГУ ТА ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ РОЗРОБКИ

1.1 Сучасні підходи до моніторингу активності персоналу

На дії співробітників впливають навколишнє середовище та чинні системи. Розглянемо, як система моніторингу впливає на тих, за ким здійснюється спостереження, і на тих, хто отримує результати моніторингу.

Наявність систем моніторингу може стримувати працівників від непродуктивної або неетичної поведінки. Водночас контроль співробітника компанії, користувача комп'ютера, через супервізора спричиняє зниження залученості, але не викликає втоми від роботи. Тому слід застосовувати інформаційну систему оцінки активності користувача в контексті високої залученості та низької втоми, щоб уникнути вигорання.

Моніторинг покращує виконання завдань, забезпечуючи зворотний зв'язок у реальному часі та дотримання робочих процесів . Аналогічно, програмне забезпечення для моніторингу продуктивності може допомогти організаціям формулювати чіткі цілі та відстежувати прогрес, сприяючи формуванню культури, орієнтованої на результат.

Однак, незважаючи на формування культури продуктивності, електронний моніторинг може перешкоджати соціальній взаємодії між лідерами та членами колективу, що призводить до зниження довіри та співпраці, а співпраця є чинником ефективності компанії.

Щоб уникнути роз'єднаності та інших негативних ефектів, організації адаптують системи моніторингу до своєї корпоративної культури та вирішують питання конфіденційності, зокрема шляхом проведення роз'яснень щодо роботи системи моніторингу. Це підвищує ймовірність досягнення позитивних результатів.

Таким чином, моніторинг спричиняє ефекти відчуження праці, формалізації відносин між виконавцями та керівниками в компанії. У випадку

застосування моніторингу активності співробітників слід зважувати всі потенційні переваги та можливі ризики.

Перераховані негативні ефекти нівелюються при індивідуальному використанні системи, оскільки при особистому застосуванні користувачі зазвичай краще контролюють ситуацію, що дозволяє адаптувати інструменти до власних потреб. Така автономія може підвищити мотивацію та концентрацію.

Моніторинг у форматі особистого використання допомагає виявляти звички, пов’язані з нераціональним використанням часу, та оптимізувати робочі процеси.

Розглянемо особливості співробітників IT-компаній порівняно з представниками інших професійних спільнот

Праця в секторі інформаційних технологій має спеціалізований характер і відрізняється від інших видів інтелектуальної діяльності. Працівники цієї сфери, зокрема розробники програмного забезпечення, інженери технічної підтримки, системні адміністратори та адміністратори баз даних, працюють у динамічному, часто кризовому середовищі, яке вимагає швидкого вирішення проблем і здатності до багатозадачності.

Співробітники IT-сфери стикаються з унікальними викликами через реактивний і різноманітний характер виконуваних завдань. Їхню роботу часто переривають, тому їм доводиться одночасно керувати кількома проєктами. Такі перерви призводять до незавершених завдань, які працівники повертаються виконувати, коли мають достатньо часу та ресурсів. Складність роботи ускладнюється ще й необхідністю ретельної організації процесів та вмінням ефективно перемикатися між задачами.

Ще одним важливим аспектом є те, як IT-фахівці систематизують інформацію на своїх комп’ютерах і повторно отримують до неї доступ. Дослідження показали, що ієрархічна система папок, хоча й широко використовується, має обмеження в організації та вилученні інформації. Багато IT-працівників надають перевагу перегляду папок для повторного доступу до даних, використовуючи пошук лише в крайніх випадках.

Співробітники IT-сфери мають високі навички роботи з комп'ютером, однак подальше вдосконалення цифрових компетенцій все ще дозволяє підвищувати ефективність праці.

Таким чином, характерними особливостями працівників IT-сфери є підвищений технологічний стрес, часте перемикання уваги, специфічна систематизація файлів на комп'ютері. Передбачається, що розроблювана інформаційна система моніторингу сприятиме зниженню технологічного стресу та уникненню надмірного перемикання між завданнями.

1.2 Особливості праці співробітників IT-сектору та проблеми техностресу

Проведемо аналіз наведених нижче систем, порівняємо їхні особливості та сформуємо нову композицію елементів в єдину інформаційну автоматизовану систему моніторингу активності.

Дані можуть збиратися з носимих пристроїв у вигляді фізіологічних показників. Носимі пристрої, такі як смарт-годинники та фітнес-трекери, надають фізіологічні дані, включаючи частоту серцевих скорочень, якість сну та рівень стресу, які можуть бути пов'язані з якістю трудового життя (WRQoL). Ці пристрої забезпечують неінвазивний спосіб моніторингу добробуту працівників, хоча можуть викликати питання конфіденційності.

Такі системи, зокрема, вимірюють рівень стресу та втоми, щоб своєчасно застосовувати відповідні дії, спрямовані на підвищення ефективності працівника шляхом покращення його фізичного та психічного стану.

Дані також можуть збиратися з комп'ютера, яким користується працівник. Інформація про цифрову взаємодію, така як активність клавіатури та миші, використання електронної пошти та взаємодія з програмами, зазвичай збирається для оцінки продуктивності та залученості. Для цього широко використовуються такі інструменти, як ActiveTrack, Time Doctor і HubStaff. Зазначені інструменти збирають дані та формують оцінки на основі статистики без застосування машинного навчання.

Окрім даних з пристроїв введення, деякі системи також збирають інформацію про мережевий трафік. Додатково до вимірювання активного часу

використання пристроїв введення, фіксуються дані про кількість написаних слів, кількість помилок у словах, кількість прокручувань коліщатка миші в різних контекстах тощо.

Також можна отримувати дані з таких джерел, як відеокамери та мікрофони. Прикладом застосування є технології комп'ютерного зору, такі як YoloV8, які використовуються для моніторингу присутності та активності працівників на робочих місцях — офісах або особистих робочих просторах у форматі віддаленої роботи. Ці системи можуть автоматично відстежувати присутність працівника на робочому місці.

Також можливо визначати не лише сам факт присутності, а й інші параметри — наприклад, емоційний стан за мімікою, тоном голосу, змістом мовлення. Збір даних із такого джерела в форматі неособистого застосування системи моніторингу часто викликає негативну реакцію у спостережуваних осіб, оскільки порушується особистий простір.

Дані також можуть надходити з датчиків, вбудованих у робоче місце. Трибоелектричні та п'єзоелектричні датчики з автономним живленням можуть відстежувати дії на робочому місці, такі як набір тексту та використання миші. Ці датчики є економічними та невибагливими в експлуатації, хоча можуть потребувати первинного налаштування та калібрування.

Датчики, що накладаються на вказівний палець, здатні вловлювати вихідні електричні сигнали під час виконання звичних офісних завдань — набору тексту, прокручування миші, написання повідомлень. Зібрані дані аналізуються за допомогою згорткових нейронних мереж (CNN) для точного розмежування видів діяльності та досягнення точності класифікації понад 98%.

Інтеграція трибоелектричних і п'єзоелектричних механізмів підвищує чутливість і продуктивність таких датчиків — наприклад, у застосунку для миші, який розпізнає різні операції за схемами напруги, забезпечуючи ефективний моніторинг поведінки користувача та збір енергії.

Крім того, п'єзоелектричні датчики застосовуються в умовах віддаленої роботи для ненав'язливого моніторингу робочого навантаження та рівня стресу

працівників, що свідчить про їхню здатність збирати довгострокові дані без створення дискомфорту.

Ці властивості підкреслюють потенціал трибоелектричних і п’єзоелектричних датчиків у вдосконаленні моніторингу робочих місць, забезпечуючи ненав’язливий, ефективний і точний засіб оцінки діяльності та добробуту IT-співробітників.

Для отримання зворотного зв’язку щодо ефективності роботи системи моніторингу можна використовувати опитування та самооцінки. Це досить традиційні методи збору даних. Опитування і самооцінки, наприклад ті, що були використані в дослідженні Pazio et al., дають уявлення про рівень фізичної активності IT-працівників. Вони показують, що значна частина співробітників повідомляла про недостатню фізичну активність під час пандемії COVID-19, і багато хто вважає, що пандемія негативно вплинула на їхній спосіб життя. Розроблювана система може розглядатися для подальшої адаптації до обставин, за яких може спостерігатися дефіцит фізичної активності.

Хоча ці методи дають суб’єктивну інформацію, вони схильні до упередженості та високого рівня відсіву. Самооцінки можуть бути упередженими — наприклад, люди часто завищують свої навички в IT-сфері через прагнення до соціальної привабливості. Це свідчить про те, що, хоча самооцінки можуть містити цінні дані, їх слід інтерпретувати з обережністю.

Натомість електронні засоби моніторингу, такі як Net Monitor for Employees, є більш об’єктивним інструментом відстеження комп’ютерної активності, дозволяючи керівникам оцінювати продуктивність і залученість без покладання виключно на дані, що надаються самими працівниками.

Існує успішний приклад поєднання самооцінок і машинного моніторингу активності для підвищення якості життя через балансування впливу роботи та відпочинку. Тому цей тип джерела інформації може бути застосований у подальших дослідженнях, коли розроблена в цій роботі автоматизована система буде використовуватися для впливу на рівень технологічного стресу та цифрового добробуту, але з певними застереженнями з огляду на зазначені

обставини.

1.3Аналіз існуючих рішень та обґрунтування необхідності нової розробки

Багато систем використовують кілька типів джерел даних: телеметрію комп’ютера, невербальну поведінку користувача комп’ютера тощо.

Деякі системи збирали дані про дії користувача і на основі ключових слів, кількості слів та подібних типів інформації робили висновки про його стан. Також використовувалася історія пошуку для прогнозування подальших дій і збору зворотного зв’язку — як прямого, так і непрямого — для налаштування поведінки програми.

У майбутньому проєкті також можна збирати історію браузера для визначення стану активності користувача залежно від його ролі. Наприклад, відвідування соціальних мереж для SMM-спеціаліста є природним, тоді як для бухгалтера це не є типовою рисою.

Також можна збирати дані зворотного зв’язку — наприклад, при блокуванні сесії для проведення перерви від комп’ютера очікувати ознаки фрустрації, такі як багаторазові натискання клавіш або різкі рухи курсора миші, що може свідчити про негативний зворотний зв’язок і неправильну оцінку активності користувача.

Використовуються навколишній звук і освітлення для визначення простою або відволікання користувача. Можна застосовувати вебкамеру або датчик освітлення для виявлення недостатнього освітлення, що може призводити до зниження ефективності. Також можна використовувати мікрофон, зокрема вбудований у вебкамеру, для визначення рівня шуму в оточенні користувача комп’ютера. Навколишній шум і розмови сусідів можуть знижувати ефективність користувача, хоча деякі типи шумів без різких звуків мають протилежний ефект.

Емоційне забарвлення контенту на комп’ютері впливає на емоційний стан користувача, що, своєю чергою, може впливати на його продуктивність. Якщо є можливість аналізувати емоційне забарвлення контенту, це можна

використовувати як один із непрямих індикаторів рівня активності користувача та як основу для оцінки його стану.

Проєктна частина «Система оцінки активності користувача ПК»

Вибір і обґрунтування методів та інструментальних засобів управління IT-проєктами

Waterfall (каскадна модель). Обрана через те, що це лінійний підхід, у якому кожна фаза проєкту (аналіз, проєктування, розробка, тестування, впровадження) виконується послідовно. Проєкт має чітко визначені вимоги, немає потреби в гнучкій зміні вимог.

Kanban**.** Обраний тому, що це візуальний метод управління завданнями, де робота переміщується по етапах виконання з одного статусу в інший, що дає наочне уявлення про стан проєкту.

GitHub**.** Обраний тому, що дозволяє безкоштовно зберігати приватний репозиторій кодової бази проєкту. Містить функціонал контролю версій проєкту.

Це технічне завдання складено на розробку «Автоматизованої системи моніторингу активності співробітників в IT-компаніях». Ця система міститиме наступний функціонал:

1. модуль аналізу списку запущених процесів;
2. модуль аналізу даних з пристроїв введення;
3. модуль розробки та зберігання документації;

2 РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ ТА ВИБІР МЕТОДІВ МАШИННОГО НАВЧАННЯ

2.1Технічне завдання на розробку системи

Технічне завдання визначає мету, призначення та основні вимоги до створення програмного комплексу «Автоматизована система моніторингу активності співробітників в IT-компаніях». Ця система призначена для збору, аналізу та класифікації даних про роботу користувачів за персональними комп’ютерами з метою підвищення ефективності праці та зниження рівня технологічного стресу.

Необхідність розробки обумовлена сучасними тенденціями цифрової трансформації, коли значна частина робочих процесів здійснюється у віртуальному середовищі. Працівники IT-сфери постійно взаємодіють із комп’ютерами, виконують багатозадачні операції, перемикаються між проєктами та стикаються з високим рівнем інформаційного навантаження. Це призводить до зниження концентрації, виникнення помилок та зростання ризику професійного вигорання. Автоматизована система моніторингу дозволяє своєчасно виявляти ознаки перевтоми, непродуктивної діяльності або відволікання від робочих завдань, а також пропонувати користувачу зробити паузу чи змінити режим роботи.

Метою створення системи є отримання готового програмного продукту, який забезпечує комплексний моніторинг активності співробітників, інтегрує дані з пристроїв введення (клавіатура, миша), аналізує перелік запущених процесів та застосовує алгоритми машинного навчання для класифікації типів діяльності. Система повинна не лише збирати інформацію, але й інтерпретувати її, формуючи висновки про продуктивність користувача та пропонуючи оптимальні рішення для підтримки його цифрового добробуту.

Призначення розробки полягає у створенні інструменту, який допомагає компаніям контролювати робочий процес без надмірного втручання у приватність співробітників. Система має бути орієнтована на підтримку балансу між ефективністю та комфортом працівника. Вона повинна працювати у

фоновому режимі, не заважаючи виконанню основних завдань, і водночас надавати керівництву узагальнені дані про рівень продуктивності команди.

До системи висуваються вимоги щодо функціональності, надійності та сумісності. Вона повинна забезпечувати можливість тимчасового блокування комп'ютера для організації перерв, формування повідомлень про необхідність паузи, а також збір і аналіз даних про взаємодію користувача з комп'ютером. Надійність системи передбачає захист від некоректних дій користувача, контроль введених даних та стабільну роботу у стандартних умовах експлуатації.

Програмне забезпечення має функціонувати на IBM-сумісних персональних комп'ютерах з архітектурою x86–x64 та мінімальним обсягом оперативної пам'яті 512 МБ. Воно повинно бути сумісним з операційними системами Windows 10–11 та підтримувати сучасні бібліотеки машинного навчання, такі як keras, sklearn та pandas. Для реалізації алгоритмів передбачається використання мов програмування C# та Python, що забезпечує гнучкість та масштабованість рішення.

Особливу увагу слід приділити програмній документації. Розроблювані модулі повинні бути самодокументованими, містити коментарі у вихідному коді та супроводжуватися довідковою системою. До складу документації мають входити технічне завдання та інструкція користувача, що забезпечить можливість подальшої підтримки та розвитку системи.

Спеціальною вимогою є створення інсталяційної версії програмного забезпечення, яка дозволить швидко розгортати систему на робочих станціях компанії. Контроль виконання здійснюється замовником, а уточнення окремих вимог можливе за домовленістю між керівником та виконавцем.

2.2. Архітектура системи та вибір технологічного стеку

Опишемо вибір оптимального варіанту реалізації штучного інтелекту для класифікації стану активності користувача ПК на основі даних з пристроїв введення.

Маємо дані наступної структури (Лістинг 1). Кожна підсесія користувача записується у вигляді структури, яка містить цільову ознаку стану активності та список дій з пристроїв введення.

Лістинг 1 – Тип даних записів з пристроїв введення

```
{
    mode: 0 | 1,
    list: [{buttonKey: number, dateTime: Timestamp}]
}
```

Ключ з кожного типу пристроїв врахований таким чином, щоб вони не перетиналися між собою — з використанням словника у Python-скрипті, відповідальному за збір цих даних. Поле mode містить набір числових значень, що позначають певний стан активності користувача. datetime є записом дати та часу.

Опишемо низку варіантів реалізації модуля класифікатора на основі такої структури даних.

Статистичний аналіз (Feature Engineering + Threshold)

Це найпростіший із розглянутих підходів — виділити ознаки (features) зі списку list і застосувати до них порогові правила.

Метод класифікації, заснований на ознаковому аналізі (feature engineering) і порогових правилах, має низку переваг, пов’язаних із його простотою та інтерпретованістю. Оскільки алгоритм базується на статистичних метриках, таких як середній і медіанний час між подіями, частотні характеристики натискань і екстремальні значення ознак, його реалізація не потребує складних обчислювальних процедур. Це робить метод ефективним для задач, де важлива швидкість обробки даних і прозорість прийняття рішень.

Крім того, завдяки використанню детермінованих правил, фахівець може легко коригувати класифікатор, спираючись на предметну область, що особливо корисно в умовах обмеженого обсягу даних або потреби у швидкому прототипуванні.

Однак цей підхід має суттєві обмеження, пов’язані з його лінійністю та залежністю від ручного налаштування. Якщо розподіл ознак для різних класів

(mode 0 і mode 1) перекривається або має складну структуру, порогові правила можуть демонструвати низьку точність. Крім того, метод не враховує можливі взаємодії між ознаками, що знижує його здатність до узагальнення. Ще одним недоліком є чутливість до викидів: такі метрики, як середнє арифметичне, можуть спотворюватися під впливом аномальних значень, що потребує додаткової попередньої обробки даних.

Статистичний підхід із пороговою класифікацією доцільно застосовувати в тих випадках, коли дані мають чітку роздільність за ключовими ознаками — чого в нашому випадку не спостерігається — а також коли критичною є швидкість роботи алгоритму, що для нас не є суворою вимогою. Цей метод особливо ефективний у ситуаціях, де допустиме ручне налаштування і де інтерпретованість моделі важливіша за її абсолютну точність — що також не є нашим пріоритетом.

Наші дані характеризуються нелінійними залежностями та високим ступенем перекриття класів, тому доцільніше використовувати методи машинного навчання. Наприклад, логістична регресія дозволяє враховувати зважений вплив кількох ознак, а дерева рішень автоматично знаходять оптимальні пороги розділення. Ми розглянемо їх далі.

У більш складних випадках, коли необхідно враховувати взаємодії між багатьма змінними, можуть застосовуватися ансамблеві алгоритми, такі як Random Forest або градієнтний бустинг (наприклад, XGBoost), які демонструють високу точність завдяки комбінуванню багатьох слабких класифікаторів — їх ми також розглянемо далі.

У нашому випадку дані містять виражені часові закономірності, що потребують аналізу послідовностей. Тому більш доречними можуть бути моделі глибокого навчання — згорткові нейронні мережі (CNN), графові (GNN) або рекурентні нейронні мережі (RNN), зокрема їхні варіанти — LSTM і GRU.

2.3 Порівняльний аналіз алгоритмів машинного навчання для класифікації активності

Кількість записів у list не повинна мати вирішального значення — початкова калібровка класифікатора має відбуватися незалежно від розміру лог-файлу. Середній/медіанний час між натисканнями (dateTime) — необхідно враховувати особливі випадки, коли користувач може працювати, уважно вивчаючи вміст екрану без активного використання пристроїв введення, і навпаки. Частота натискань (наприклад, скільки разів buttonKey зустрічається в певному діапазоні) — також слід враховувати особливі випадки, як і для попередніх ознак, наприклад, при багаторазовій прокрутці коліщатка миші вниз або використанні комбінацій клавіш типу Alt+Tab.

Класифікація. Якщо list порожній — mode дорівнює значенню за замовчуванням. Якщо середній час між натисканнями менший за заданий поріг — mode дорівнює певному значенню. Якщо частота натискань певної кнопки перевищує поріг — mode дорівнює іншому значенню.

Нижче наведено приклад реалізації з використанням статистичного підходу з пороговою класифікацією (Лістинг 2). Рішення витягує 6 ключових ознак зі списку подій:

1. count — загальна кількість натискань;
2. mean_time_diff — середній час між натисканнями (мс);
3. median_time_diff — медіанний час (стійкий до викидів);
4. min_key, max_key — діапазон значень кнопок;
5. freq_low — частка натискань кнопок з buttonKey < 5 (приклад категоризації).

Часові мітки можуть надходити в довільному порядку. Сортування гарантує коректне обчислення інтервалів. Передбачено перевірки, наприклад, захист від помилок, якщо list порожній. Використовуються медіани, оскільки медіана стійка до викидів (наприклад, якщо було одне дуже довге натискання). Використовується ознака freq_low через припущення, що велика кількість натискань на кнопки з малими номерами може вказувати на певне значення цільової ознаки.

Можна було додати стандартне відхилення часових інтервалів (метрика «розкиду»). Можна було рахувати частоту конкретних кнопок (наприклад, buttonKey = 3), але це ускладнило б модель без очевидної потреби.

Пояснення правил класифікації:

Якщо count < 3 → mode = 0 (мало даних — консервативний варіант).

Якщо mean_time_diff > 1000 або median_time_diff > 800 → mode = 1 (великі інтервали — можливо, mode = 1).

Якщо freq_low > 0.7 → mode = 0 (понад 70% натискань на кнопки з малими номерами — схиляємося до mode = 0).

Якщо max_key - min_key > 10 → mode = 1 (дуже різні кнопки — можливо, mode = 1).

Лістинг 2 – Приклад реалізації статистичного підходу з пороговою класифікацією

```python
import time
import tracemalloc
from typing import List, Dict, Union

import numpy as np
from sklearn.metrics import classification_report

from device_input.device_log_loader import load_device_logs


def extract_features(data: Dict[str, Union[int, List[Dict[str, int]]]]) -> Dict[str, float]:
    """
    Видобуває статистичні ознаки з вхідних даних для подальшої класифікації.

    Args:
        data: Вхідні дані у форматі {'mode': int, 'list': [{'buttonKey': int, 'dateTime': int}, ...]}

    Returns:
        Словник з обчисленими ознаками:
        - count: кількість записів
        - mean_time_diff: середній час між натисканнями (у секундах)
        - median_time_diff: медіанний час між натисканнями
        - min_key: мінімальне значення buttonKey
        - max_key: максимальне значення buttonKey
        - freq_low: частка натискань з buttonKey < 5
    """
    events = data['list']
    timestamps = sorted([e['dateTime'] for e in events])
    button_keys = [e['buttonKey'] for e in events]

    # Обчислення часових інтервалів між подіями
    time_diffs = np.diff(timestamps) if len(timestamps) > 1 else [0]

    features = {
        'count': len(events),
        'mean_time_diff': np.mean(time_diffs),
        'median_time_diff': np.median(time_diffs),
        'min_key': min(button_keys) if button_keys else 0,
        'max_key': max(button_keys) if button_keys else 0,
        'freq_low': sum(k < 5 for k in button_keys) / len(button_keys) if button_keys else 0
    }

    return features


def classify_by_thresholds(features: Dict[str, float]) -> int:
```

```
    """
    Класифікує mode на основі витягнених ознак та порогових правил.

    Args:
        features: Словник з витягненими ознаками

    Returns:
        Передбачене значення mode (0 або 1)
    """
    # Емпірично визначені порогові значення
    if features['count'] < 3:
        return 0

    if (features['mean_time_diff'] > 1000 or
            features['median_time_diff'] > 800):
        return 1

    if features['freq_low'] > 0.7:
        return 0

    if features['max_key'] - features['min_key'] > 10:
        return 1

    # За замовчуванням повертаємо 0
    return 0


# Приклад використання
if __name__ == "__main__":
    # Приклад вхідних даних
    sample_data = load_device_logs(1000)

    # Вимірювання використання пам'яті до виконання
    tracemalloc.start()
    start_inf = time.time()

    # Передбачення на тестових даних із заміром часу
    y_pred = []
    for item in sample_data:
        # Видобуваємо ознаки
        features = extract_features(item)
        # Класифікуємо
        y_pred.append(classify_by_thresholds(features))

    end_inf = time.time()
    inference_time = end_inf - start_inf

    # Вимірювання пам'яті після виконання
    current, peak = tracemalloc.get_traced_memory()
    max_ram_usage = peak / (1024 ** 2)  # у MB
    tracemalloc.stop()

```

```
    # Справжні значення mode
    y_test = [sample_data['mode'] for sample_data in sample_data]

    print(classification_report(y_test, y_pred, zero_division=0))
    print(f"Max RAM Usage: {max_ram_usage:.2f} MB")
    print(f"Inference time: {inference_time:.4f} s")
```

Класичне машинне навчання (ML)

Далі розглянемо кілька класичних моделей: випадковий ліс (Random Forest), XGBoost, CatBoost, логістична регресія (Logistic Regression).

Логістична регресія

Логістична регресія є статистичним методом класифікації, що має низку переваг у контексті запропонованої задачі. Однією з ключових переваг є висока інтерпретованість моделі. Оскільки логістична регресія обчислює вагові коефіцієнти для кожної ознаки, це дозволяє аналітику оцінити внесок окремих факторів — таких як частота натискань певних клавіш або дисперсія часових інтервалів — у визначення цільової ознаки mode.

З обчислювальної точки зору метод є досить ефективним: алгоритм швидко сходиться навіть на відносно великих вибірках, а час отримання інференсу (тобто кінцевого результату обробки нових даних) мінімальний, що робить його придатним для систем реального часу.

Крім того, логістична регресія демонструє стабільну роботу в умовах лінійної роздільності класів. Якщо витягнуті ознаки дійсно містять лінійно розділяючі закономірності (що, на жаль, ми не можемо гарантувати у нашому випадку), модель досягне задовільної точності без перенавчання — особливо при використанні регуляризації L1 або L2.

Регуляризація обмежує величину вагових коефіцієнтів моделі, додаючи штраф до функції втрат. У L1-регуляризації використовується сума абсолютних значень ваг, а в L2 — сума квадратів вагових значень.

Обмеження методу

Метод має суттєві обмеження, пов’язані з його лінійною природою. Якщо залежність між ознаками та цільовою змінною є нелінійною (наприклад, якщо

ознака mode визначається складними комбінаційними або часовими патернами), передбачувальна здатність моделі може виявитися недостатньою.

Крім того, логістична регресія вимагає ретельного проєктування ознакового простору. Невдалий вибір або відсутність інформативних ознак — таких як часові похідні або частотні метрики — призведе до зниження якості класифікації.

Ще одним обмеженням є чутливість моделі до дисбалансу класів. При значному переважанні одного з класів (наприклад, якщо mode = 0 зустрічається у 90% випадків) алгоритм схильний до зміщення в бік мажоритарного класу. Для корекції цього ефекту необхідно застосовувати методи балансування — такі як зважування класів або синтетичне збільшення вибірки міноритарного класу (SMOTE).

Логістична регресія також не враховує потенційні часові залежності між спостереженнями всередині списку list. Хоча такі ознаки, як середній інтервал між подіями або дисперсія часу, частково відображають часову структуру, вони не здатні моделювати складні послідовні закономірності, для яких більш доречними були б рекурентні нейронні мережі (RNN) або трансформери.

У випадках мультиколінеарності ознак — тобто коли спостерігається сильна кореляція між ознаками, наприклад, між «кількістю натискань кнопки 5» і «часткою натискань кнопки 5» — оцінки коефіцієнтів регресії стають нестабільними. Це може вимагати додаткової обробки даних, такої як відбір ознак або застосування методів зменшення розмірності.

Логістична регресія є виправданим базовим рішенням, якщо попередній аналіз даних підтверджує лінійну роздільність класів, а також якщо інтерпретованість моделі є критично важливою вимогою. Таким чином, логістична регресія не є найкращим варіантом для нашого випадку.

Нижче наведено приклад реалізації рішення з використанням логістичної регресії (Лістинг 3). Опишемо представлене рішення.

Відбувається перетворення сирих даних (послідовності натискань) у числові ознаки. Обчислюються наступні характеристики:

1. session_length — довжина сесії, оскільки вона може корелювати з класом; min/max не використовуються через нестачу інформації;
2. time_mean — середній час між натисканнями, як важливий патерн;
3. btn_ratio — відносна частота певної кнопки як можливий маркер;
4. rapid_clicks — кількість швидких натискань як потенційно значуща ознака.

Використовуються **відносні частоти** для ознаки btn_ratio, а не абсолютні значення counts, оскільки це дозволяє моделі не залежати від довжини сесії.

Додані **часові характеристики**, оскільки логістична регресія не здатна працювати з послідовностями без попередньої обробки — тобто з послідовностями змінної довжини та часовими залежностями між елементами.

Для попередньої обробки використовується **StandardScaler**, оскільки логістична регресія чутлива до масштабу ознак, і це необхідно, коли ознаки мають різні одиниці вимірювання — наприклад, час і частота.

Параметр class_weight встановлено як 'balanced', оскільки в даних може бути дисбаланс — наприклад, співвідношення класів 70 на 30. Це дозволяє моделі компенсувати нерівномірність розподілу, автоматично коригуючи ваги класів.

Параметр penalty дорівнює 'l2', оскільки цей варіант зазвичай працює краще, ніж 'l1' для «гладких» даних — тобто таких, що мають невеликі, поступові зміни, не містять різких викидів або стрибків, корелюють між собою плавно. L2-регуляризація штрафує великі ваги, але не обнуляє їх повністю (на відміну від L1). Це добре працює, коли всі ознаки вносять невеликий внесок і між ними існує кореляція.

Параметр max_iter дорівнює 1000, оскільки для складних даних стандартних 100 ітерацій може бути недостатньо для збіжності моделі.

Лістинг 3 – Реалізація рішення з використанням логістичної регресії

```python
import time
import tracemalloc
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

from device_input.device_log_loader import load_device_logs

class ButtonPatternClassifier:
    def __init__(self):
        """Ініціалізація пайплайна з масштабуванням та логістичною регресією"""
        self.model = make_pipeline(
            StandardScaler(),
            LogisticRegression(
                penalty='l2',
                C=1.0,
                solver='lbfgs',
                class_weight='balanced',
                max_iter=1000,
                random_state=42
            )
        )

    def prepare_dataset(self, json_data):
        X = []
        y = []

        for item in json_data:
            X.append(item['list'])
            y.append(item['mode'])

        return X, y

    def _extract_features(self, dataset):
        """Витягнення ознак із сирих даних"""
        features = []
        for events in dataset:
            # Базові статистичні характеристики
            num_events = len(events)
            button_keys = [e['buttonKey'] for e in events]
            timestamps = [e['dateTime'] for e in events]

            # Часові характеристики
            time_diffs = []
            if len(timestamps) > 1:
                time_diffs = [
                    (timestamps[i] - timestamps[i + 1])
                    for i in range(len(timestamps) - 1)
                ]

                # Частотні характеристики кнопок
```

```
            # Частотні характеристики кнопок
            button_counts = {i: 0 for i in range(1, 81)}
            for btn in button_keys:
                if btn in button_counts:
                    button_counts[btn] += 1

            # Формування вектора ознак
            feature_vec = {
                'session_length': num_events,
                'time_mean': np.mean(time_diffs) if time_diffs else 0,
                'time_std': np.std(time_diffs) if time_diffs else 0,
                **{f'btn{i}_ratio': button_counts[i] / num_events for i in range(1, 81)},
                'rapid_clicks': sum(1 for diff in time_diffs if diff < 1.0)
            }
            features.append(feature_vec)

        return pd.DataFrame(features)

    def fit(self, X, y):
        """Навчання моделі на розмічених даних"""
        X_features = self._extract_features(X)
        self.model.fit(X_features, y)
        return self

    def predict(self, X):
        """Передбачення класів для нових даних"""
        X_features = self._extract_features(X)
        return self.model.predict(X_features)

    def predict_proba(self, X):
        """Передбачення ймовірностей класів"""
        X_features = self._extract_features(X)
        return self.model.predict_proba(X_features)

    def evaluate(self, X, y):
        """Оцінка якості моделі"""
        X_features = self._extract_features(X)
        y_pred = self.model.predict(X_features)
        return classification_report(y, y_pred, output_dict=True)

# Приклад використання
if __name__ == "__main__":
    sample_data = load_device_logs(1000)
    classifier = ButtonPatternClassifier()
    X_data, y_labels = classifier.prepare_dataset(sample_data)
    X_train, X_test, y_train, y_test = train_test_split(X_data, y_labels, test_size=0.2, random_state=42)

    # Вимірювання використання пам'яті до навчання
    tracemalloc.start()
    start_train = time.time()
    classifier.fit(X_train, y_train)
    end_train = time.time()
    training_time = end_train - start_train
```

```
    # Вимірювання пам'яті після навчання
    current, peak = tracemalloc.get_traced_memory()
    max_ram_usage = peak / (1024 ** 2)  # у MB
    tracemalloc.stop()

    # Оцінка якості
    sample_data = X_test
    start_inf = time.time()
    y_pred = classifier.predict(sample_data)
    end_inf = time.time()
    inference_time = end_inf - start_inf

    print(classification_report(y_test, y_pred))
    print(f"Max RAM Usage: {max_ram_usage:.2f} MB")
    print(f"Inference time: {inference_time:.4f} s")
```

Випадковий ліс (Random Forest)

Цей підхід має високу інтерпретованість, оскільки витягнуті ознаки (такі як кількість записів, часові інтервали між подіями та частота натискань кнопок) мають чітку семантику. Це дозволяє аналізувати внесок кожної ознаки у передбачення цільової змінної mode.

Random Forest також забезпечує оцінку важливості ознак (feature importance), що допомагає виявити найбільш значущі фактори, які впливають на класифікацію. Це може бути корисним для подальших досліджень, хоча не є критично необхідним для поточного етапу.

Алгоритм демонструє стійкість до перенавчання завдяки ансамблевому характеру роботи: агрегування передбачень множини дерев знижує ймовірність помилки. Крім того, Random Forest ефективно працює з різнорідними даними — як числовими (часові мітки, статистики), так і категоріальними (ідентифікатори кнопок) — без потреби у складній попередній обробці.

Метод автоматично враховує взаємодії ознак завдяки побудові множини рішень дерев, кожне з яких може виявляти нелінійні залежності. Наприклад, модель може виявити, що комбінація частого натискання кнопки 5 і низької дисперсії часових інтервалів корелює з mode = 1.

З точки зору масштабованості, після навчання модель швидко обробляє нові дані, оскільки передбачення вимагає лише одноразового обчислення ознак і проходу по ансамблю дерев.

3.ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1. Реалізація модулів збору даних та попередньої обробки

Ефективність моделі значною мірою залежить від обсягу та репрезентативності даних. У випадку невеликої кількості навчальних прикладів або сильного дисбалансу класів (наприклад, якщо один mode зустрічається значно частіше за інший), якість класифікації може знижуватися через зміщене навчання.

Для боротьби з дисбалансом класів можна застосовувати методи ресемплінгу:

1. **Oversampling** — доповнення даних для міноритарного класу;
2. **Undersampling** — зменшення даних мажоритарного класу;
3. **Зважування класів** — у функції втрат.

Ключовим обмеженням є необхідність ручної розробки ознак (feature engineering). Якість передбачень безпосередньо залежить від того, наскільки вдало обрані та сконструйовані ознаки. Наприклад, якщо цільова змінна залежить від складних часових патернів (таких як певні послідовності натискань кнопок), модель може не виявити ці залежності без явного кодування відповідних ознак.

Важливим недоліком є ігнорування часової послідовності подій. Оскільки Random Forest оперує агрегованими статистиками, він не враховує порядок елементів у list. Якщо mode залежить від послідовності натискань (наприклад, «кнопка 1, потім кнопка 2»), буде потрібна додаткова розробка ознак або перехід до методів, спеціалізованих на часових рядах.

Ще однією проблемою може бути розрідженість категоріальних ознак. Рідко зустрічаються значення buttonKey (наприклад, унікальні або малоповторювані кнопки) можуть не мати значного впливу на модель, що потребуватиме їх групування або використання альтернативних методів кодування.

При великих обсягах даних обчислення агрегованих ознак (таких як дисперсія часових інтервалів) може бути обчислювально затратним, що уповільнить як навчання, так і передбачення.

Для підвищення якості класифікації доцільно експериментувати з додатковими ознаками, такими як порядкові характеристики (наприклад, перша або остання натиснута кнопка) та комбінаційні патерни (наприклад, частота певних послідовностей кнопок), що призведе до трудомісткого перебору списку оптимальних ознак, який може змінюватися від користувача до користувача.

Оскільки ознака mode залежить від часових залежностей, розглядаємо дану модель як не найкращий варіант порівняно з наступними.

Нижче наведено приклад реалізації рішення з використанням Random Forest (Лістинг 4). Опишемо реалізацію.

Витягнуті ознаки:

1. count — загальна кількість подій, показник активності, може корелювати з mode;
2. time_diff_mean і time_diff_std — показують «ритм» взаємодій, наприклад, якщо є швидкі кліки, тоді mode = 1;
3. time_diff_max і time_diff_min — виявляють аномалії, наприклад, якщо є довгі паузи, можливо, mode = 0;
4. абсолютні (key_{i}_count) і відносні (key_{i}_ratio) частоти кнопок — обмеження до x кнопок (range(1,x)) через припущення про обмежену кількість елементів множини кнопок;
5. fast_events — фіксує «бурні» сесії, наприклад, коли більше 3 кліків за 5 секунд, тоді mode = 1.

Параметри моделі:

1. n_estimators = 100 — цього достатньо для більшості задач;
2. max_depth = 10 — обмеження складності дерев;
3. class_weight = 'balanced' — оскільки один mode може зустрічатися частіше за інший.

Лістинг 4 – Реалізація рішення з використанням Random Forest

```
import time
import tracemalloc
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

from device_input.device_log_loader import load_device_logs

class ButtonPatternClassifier:
    def __init__(self):
        """Ініціалізація пайплайна з масштабуванням та логістичною регресією"""
        self.model = make_pipeline(
            StandardScaler(),
            LogisticRegression(
                penalty='l2',
                C=1.0,
                solver='lbfgs',
                class_weight='balanced',
                max_iter=1000,
                random_state=42
            )
        )

    def prepare_dataset(self, json_data):
        X = []
        y = []

        for item in json_data:
            X.append(item['list'])
            y.append(item['mode'])

        return X, y

    def _extract_features(self, dataset):
        """Витягнення ознак із сирих даних"""
        features = []
        for events in dataset:
            # Базові статистичні характеристики
            num_events = len(events)
            button_keys = [e['buttonKey'] for e in events]
            timestamps = [e['dateTime'] for e in events]

            # Часові характеристики
            time_diffs = []
            if len(timestamps) > 1:
                time_diffs = [
                    (timestamps[i] - timestamps[i + 1])
                    for i in range(len(timestamps) - 1)
                ]

```

```
            # Частотні характеристики кнопок
            button_counts = {i: 0 for i in range(1, 81)}
            for btn in button_keys:
                if btn in button_counts:
                    button_counts[btn] += 1

            # Формування вектора ознак
            feature_vec = {
                'session_length': num_events,
                'time_mean': np.mean(time_diffs) if time_diffs else 0,
                'time_std': np.std(time_diffs) if time_diffs else 0,
                **{f'btn{i}_ratio': button_counts[i] / num_events for i in range(1, 81)},
                'rapid_clicks': sum(1 for diff in time_diffs if diff < 1.0)
            }
            features.append(feature_vec)

        return pd.DataFrame(features)

    def fit(self, X, y):
        """Навчання моделі на розмічених даних"""
        X_features = self._extract_features(X)
        self.model.fit(X_features, y)
        return self

    def predict(self, X):
        """Передбачення класів для нових даних"""
        X_features = self._extract_features(X)
        return self.model.predict(X_features)

    def predict_proba(self, X):
        """Передбачення ймовірностей класів"""
        X_features = self._extract_features(X)
        return self.model.predict_proba(X_features)

    def evaluate(self, X, y):
        """Оцінка якості моделі"""
        X_features = self._extract_features(X)
        y_pred = self.model.predict(X_features)
        return classification_report(y, y_pred, output_dict=True)

# Приклад використання
if __name__ == "__main__":
    sample_data = load_device_logs(1000)
    classifier = ButtonPatternClassifier()
    X_data, y_labels = classifier.prepare_dataset(sample_data)
    X_train, X_test, y_train, y_test = train_test_split(X_data, y_labels, test_size=0.2, random_state=42)

    # Вимірювання використання пам'яті до навчання
    tracemalloc.start()
    start_train = time.time()
    classifier.fit(X_train, y_train)
    end_train = time.time()
    training_time = end_train - start_train
```

```
    # Вимірювання пам'яті після навчання
    current, peak = tracemalloc.get_traced_memory()
    max_ram_usage = peak / (1024 ** 2)  # у MB
    tracemalloc.stop()

    # Оцінка якості
    sample_data = X_test
    start_inf = time.time()
    y_pred = classifier.predict(sample_data)
    end_inf = time.time()
    inference_time = end_inf - start_inf

    print(classification_report(y_test, y_pred))
    print(f"Max RAM Usage: {max_ram_usage:.2f} MB")
    print(f"Inference time: {inference_time:.4f} s")
```

Випадковий ліс (Random Forest)

Цей підхід має високу інтерпретованість, оскільки витягнуті ознаки (такі як кількість записів, часові інтервали між подіями та частота натискань кнопок) мають чітку семантику. Це дозволяє аналізувати внесок кожної ознаки у передбачення цільової змінної mode.

Random Forest також забезпечує оцінку важливості ознак (feature importance), що допомагає виявити найбільш значущі фактори, які впливають на класифікацію. Це може бути корисним для подальших досліджень, хоча не є критично необхідним для поточного етапу.

Алгоритм демонструє стійкість до перенавчання завдяки ансамблевому характеру роботи: агрегування передбачень множини дерев знижує ймовірність помилки. Крім того, Random Forest ефективно працює з різнорідними даними — як числовими (часові мітки, статистики), так і категоріальними (ідентифікатори кнопок) — без потреби у складній попередній обробці.

Метод автоматично враховує взаємодії ознак завдяки побудові множини рішень дерев, кожне з яких може виявляти нелінійні залежності. Наприклад, модель може виявити, що комбінація частого натискання кнопки 5 і низької дисперсії часових інтервалів корелює з mode = 1.

З точки зору масштабованості, після навчання модель швидко обробляє нові дані, оскільки передбачення вимагає лише одноразового обчислення ознак і проходу по ансамблю дерев.

Обмеження методу

Ефективність моделі значною мірою залежить від обсягу та репрезентативності даних. У випадку невеликої кількості навчальних прикладів або сильного дисбалансу класів (наприклад, якщо один mode зустрічається значно частіше за інший), якість класифікації може знижуватися через зміщене навчання.

Для боротьби з дисбалансом класів можна застосовувати методи ресемплінгу:

- Oversampling — доповнення даних для міноритарного класу;

- Undersampling — зменшення даних мажоритарного класу;
- Зважування класів — у функції втрат.

Ключовим обмеженням є необхідність ручної розробки ознак (feature engineering). Якість передбачень безпосередньо залежить від того, наскільки вдало обрані та сконструйовані ознаки. Наприклад, якщо цільова змінна залежить від складних часових патернів (таких як певні послідовності натискань кнопок), модель може не виявити ці залежності без явного кодування відповідних ознак.

Важливим недоліком є ігнорування часової послідовності подій. Оскільки Random Forest оперує агрегованими статистиками, він не враховує порядок елементів у list. Якщо mode залежить від послідовності натискань (наприклад, «кнопка 1, потім кнопка 2»), буде потрібна додаткова розробка ознак або перехід до методів, спеціалізованих на часових рядах.

Ще однією проблемою може бути розрідженість категоріальних ознак. Рідко зустрічаються значення buttonKey (наприклад, унікальні або малоповторювані кнопки) можуть не мати значного впливу на модель, що потребуватиме їх групування або використання альтернативних методів кодування.

При великих обсягах даних обчислення агрегованих ознак (таких як дисперсія часових інтервалів) може бути обчислювально затратним, що уповільнить як навчання, так і передбачення.

Для підвищення якості класифікації доцільно експериментувати з додатковими ознаками, такими як порядкові характеристики (наприклад, перша або остання натиснута кнопка) та комбінаційні патерни (наприклад, частота певних послідовностей кнопок), що призведе до трудомісткого перебору списку оптимальних ознак, який може змінюватися від користувача до користувача.

Оскільки ознака mode залежить від часових залежностей, розглядаємо дану модель як не найкращий варіант порівняно з наступними.

Нижче наведено приклад реалізації рішення з використанням Random Forest (Лістинг 4). Опишемо реалізацію.

Витягнуті ознаки:

1 count — загальна кількість подій, показник активності, може корелювати з mode;

2 time_diff_mean і time_diff_std — показують «ритм» взаємодій, наприклад, якщо є швидкі кліки, тоді mode = 1;

3 time_diff_max і time_diff_min — виявляють аномалії, наприклад, якщо є довгі паузи, можливо, mode = 0;

4 абсолютні (key_{i}_count) і відносні (key_{i}_ratio) частоти кнопок — обмеження до x кнопок (range(1,x)) через припущення про обмежену кількість елементів множини кнопок;

5 fast_events — фіксує «бурні» сесії, наприклад, коли більше 3 кліків за 5 секунд, тоді mode = 1.

Параметри моделі:

- n_estimators = 100 — цього достатньо для більшості задач;
- max_depth = 10 — обмеження складності дерев;
- class_weight = 'balanced' — оскільки один mode може зустрічатися частіше за інший.

Лістинг 5 – Реалізація рішення з використанням Random Forest

```python
import time
import tracemalloc
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from device_input.device_log_loader import load_device_logs

def extract_features(data):
    """Витягує ознаки зі списку подій"""
    features = {}
    # Базові ознаки
    features['count'] = len(data)

    # Часові ознаки
    timestamps = [x['dateTime'] for x in data]
    time_diffs = np.diff(timestamps)

    if len(time_diffs) > 0:
        features['time_diff_mean'] = np.mean(time_diffs)
        features['time_diff_std'] = np.std(time_diffs)
        features['time_diff_max'] = np.max(time_diffs)
        features['time_diff_min'] = np.min(time_diffs)
    else:
        features.update({
            'time_diff_mean': 0,
            'time_diff_std': 0,
            'time_diff_max': 0,
            'time_diff_min': 0
        })

    # Частотні ознаки
    button_keys = [x['buttonKey'] for x in data]
    unique_keys, counts = np.unique(button_keys, return_counts=True)
    key_counts = dict(zip(unique_keys, counts))

    # Додаємо кількість натискань для кожної кнопки (до 10)
    for i in range(1, 11):
        features[f'key_{i}_count'] = key_counts.get(i, 0)

    # Відносні частоти
    total_presses = len(button_keys)
    for i in range(1, 11):
        features[f'key_{i}_ratio'] = (
            features[f'key_{i}_count'] / total_presses if total_presses > 0 else 0
        )

    # Часові патерни
    if len(time_diffs) > 0:
        features['fast_events_5s'] = np.sum(np.array(time_diffs) <= 5)
        features['fast_events_10s'] = np.sum(np.array(time_diffs) <= 10)
```

```
    else:
        features.update({
            'fast_events_5s': 0,
            'fast_events_10s': 0
        })

    return features


def prepare_dataset(json_data):
    """Готує датасет із сирих JSON-даних"""
    X = []
    y = []

    for item in json_data:
        # Витягуємо цільову ознаку
        y.append(item['mode'])

        # Витягуємо ознаки зі списку подій
        features = extract_features(item['list'])
        X.append(features)

    # Перетворюємо у DataFrame
    feature_df = pd.DataFrame(X)
    return feature_df, np.array(y)


# Приклад використання
if __name__ == "__main__":
    sample_data = load_device_logs(1000)

    # Підготовка даних
    X, y = prepare_dataset(sample_data)

    # Розбиття на train/test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    # Вимірювання використання пам'яті до навчання
    tracemalloc.start()
    start_train = time.time()

    # Навчання моделі
    model = RandomForestClassifier(
        n_estimators=100,
        max_depth=10,
        random_state=42,
        class_weight='balanced'  # Для несбалансованих даних
    )
    model.fit(X_train, y_train)

    end_train = time.time()
    training_time = end_train - start_train
```

```
105
106     # Вимірювання пам'яті після навчання
107     current, peak = tracemalloc.get_traced_memory()
108     max_ram_usage = peak / (1024 ** 2)  # у MB
109     tracemalloc.stop()
110
111     # Передбачення на тестових даних з вимірюванням часу
112     start_inf = time.time()
113     y_pred = model.predict(X_test)
114     end_inf = time.time()
115     inference_time = end_inf - start_inf
116
117     # Оцінка якості
118     print(classification_report(y_test, y_pred))
119     print(f"Max RAM Usage: {max_ram_usage:.2f} MB")
120     print(f"Inference time: {inference_time:.4f} s")
```

Градієнтний бустинг

Однією з переваг цього підходу, як і деяких інших, розглянутих у даній роботі, є висока інтерпретованість ознак. Оскільки ознаки формуються на основі явних статистичних і часових метрик (таких як кількість записів, різниця між часовими мітками, частота натискань кнопок), їхній вплив на модель можна проаналізувати за допомогою методів оцінки важливості ознак (feature importance). Це дозволяє не лише вдосконалювати модель, а й розуміти, які патерни в даних є найбільш значущими для класифікації.

Іншою важливою перевагою є ефективність роботи алгоритмів бустингу на відносно невеликих обсягах даних. На відміну від глибокого навчання, яке потребує значних обчислювальних ресурсів і великих датасетів, методи на кшталт XGBoost і CatBoost демонструють високу точність навіть на вибірках розміром у тисячі прикладів.

Крім того, ці алгоритми добре справляються з різнотипними ознаками, включаючи категоріальні змінні (наприклад, buttonKey), що робить їх універсальним інструментом для задач класифікації.

Швидкість навчання і передбачення також є суттєвим плюсом. Бустингові моделі оптимізовані для швидкої обробки даних і потребують менше обчислювальних потужностей порівняно з нейромережевими архітектурами. Це особливо важливо у сценаріях, де необхідне оперативне розгортання або частий ретроспективний аналіз.

Нарешті, вбудовані механізми регуляризації в XGBoost і CatBoost знижують ризик перенавчання, що критично при роботі з обмеженими або зашумленими даними.

Обмеження методу

Основним обмеженням є залежність від ручного конструювання ознак (feature engineering). Оскільки модель працює лише з агрегованими статистиками, якість класифікації безпосередньо залежить від того, наскільки вдало обрані та сконструйовані ознаки. Це потребує глибокого розуміння предметної області і може бути трудомістким процесом, особливо якщо в даних присутні складні часові або послідовнісні залежності.

Ще одним обмеженням є неспроможність класичних ML-методів ефективно враховувати порядок елементів у послідовності. У той час як нейромережеві підходи (наприклад, RNN або Transformer) здатні аналізувати часові ряди з урахуванням їхньої структури, бустингові моделі оперують лише агрегованими характеристиками. Це може призвести до втрати важливих патернів, особливо якщо цільовий показник (mode) залежить від динаміки натискань, а не лише від їхніх сумарних статистик.

Проблеми також можуть виникати при роботі з розрідженими даними. Наприклад, якщо змінна buttonKey приймає багато унікальних значень, частотні ознаки можуть стати надто розрідженими, що негативно позначиться на здатності моделі до узагальнення.

Крім того, якщо розмітка даних містить шум або є незбалансованою, це може призвести до зміщення передбачень і погіршення метрик якості.

Нарешті, класичні ML-моделі погано адаптуються до змін у розподілі даних — явища concept drift (коли відбувається еволюція даних, яка робить модель неактуальною). Якщо з часом змінюється поведінка користувача (наприклад, зростає частота натискань певних кнопок), модель потребуватиме періодичного перенавчання на актуальних даних.

Вибір алгоритму градієнтного бустингу

Серед сучасних реалізацій розглянемо CatBoost, XGBoost і LightGBM:

- CatBoost демонструє виняткову ефективність при роботі з категоріальними ознаками завдяки вбудованим механізмам їхньої обробки. Алгоритм використовує інноваційний підхід ordered boosting, який мінімізує зміщення оцінок через спеціальну схему перестановок. Ця особливість робить CatBoost особливо цінним у задачах, де переважають категоріальні змінні або коли потрібна мінімальна попередня обробка даних. Крім того, CatBoost показує стабільні результати на відносно невеликих вибірках і має ефективну реалізацію з підтримкою GPU-прискорення.
- XGBoost (eXtreme Gradient Boosting) зберігає провідні позиції у задачах, що потребують максимальної точності передбачень. Його ключова перевага — комплексна система регуляризації, яка включає L1 (lasso) і L2 (ридж) нормалізацію, що значно знижує ризик перенавчання. Алгоритм підтримує різні типи цільових функцій і забезпечує чудову масштабованість завдяки механізмам розподілених обчислень. Однак слід зазначити, що XGBoost потребує більш ретельної підготовки категоріальних ознак порівняно з CatBoost.
- LightGBM (Light Gradient Boosting Machine) оптимізований для обробки великомасштабних даних. В його основі лежать два ключові алгоритмічні вдосконалення: Gradient-based One-Side Sampling (GOSS) для ефективного відбору об’єктів і Exclusive Feature Bundling (EFB) для оптимального використання ознакового простору. Ці інновації дозволяють LightGBM значно перевершувати конкурентів за швидкістю навчання при роботі з великими обсягами даних, хоча це може супроводжуватися дещо меншою стабільністю на малих вибірках.

Нижче наведено приклад реалізації рішення з використанням градієнтного бустингу XGBoost (Лістинг 5). Далі описано реалізацію

Витягнуті ознаки. Ознака count — загальна кількість натискань, важлива, оскільки різні значення mode можуть відповідати різному рівню активності.

Здійснюється нормалізація за загальною кількістю, що дозволяє зробити ознаки порівнюваними між сесіями різної довжини. Ознака unique_buttons — кількість унікальних кнопок, показує різноманітність взаємодії. Відбувається динамічне обчислення ознак для кожної кнопки — button_X_freq, що дозволяє моделі виявляти важливі комбінації. Додатково обчислюються середнє значення (mean) — загальний темп взаємодії, стандартне відхилення (std) — нерівномірність активності, екстремальні значення (max і min) — вказують на викиди в поведінці, медіана (median) — стійка до викидів оцінка темпу. Ознака rapid_clicks, як і в інших рішеннях, визначає поведінковий патерн частих натискань

У попередній обробці даних відбувається поступове накопичення ознак через pd.concat — це ефективніше, ніж попереднє створення масиву. Параметр ignore_index=True гарантує послідовну нумерацію, а операція fillna(0) обробляє випадки з одним натисканням, коли часові різниці не обчислюються

Параметри моделі. Параметр n_estimators=100 — забезпечує баланс між часом навчання та якістю. Параметр max_depth=5 — запобігає перенавчанню при збереженні виразності. Параметри subsample і colsample_bytree — забезпечують додаткову регуляризацію. Параметр eval_metric='logloss' — оптимальний для задач бінарної класифікації

Лістинг 6 – Реалізація моделі XGBoost.

```python
import time
import tracemalloc

import numpy as np
import pandas as pd
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

from device_input.device_log_loader import load_device_logs


# Функция для извлечения признаков из сырых данных
def extract_features(data):
    """Перетворює сирі дані у DataFrame з ознаками"""
    features = {
        'count': len(data),  # Загальна кількість натискань
        'unique_buttons': len(set(d['buttonKey'] for d in data)),  # Унікальні кнопки
    }

    # Частотні ознаки для кнопок
    button_counts = {}
    for d in data:
        button = d['buttonKey']
        button_counts[button] = button_counts.get(button, 0) + 1

    # Часові характеристики
    timestamps = [d['dateTime'] for d in data]  # Використовуємо часові мітки напряму
    timestamps.sort()

    if len(timestamps) > 1:
        time_diffs = [
            timestamps[i + 1] - timestamps[i]
            for i in range(len(timestamps) - 1)
        ]

        features.update({
            'time_mean': np.mean(time_diffs),
            'time_std': np.std(time_diffs),
            'time_max': max(time_diffs),
            'time_min': min(time_diffs),
            'time_median': np.median(time_diffs)
        })

        features['rapid_clicks'] = sum(1 for diff in time_diffs if diff < 2)
    else:
        # Значення за замовчуванням, якщо недостатньо часових міток
        features.update({
            'time_mean': 0,
            'time_std': 0,
            'time_max': 0,
            'time_min': 0,
            'time_median': 0,
            'rapid_clicks': 0
```

```
        })

    return pd.DataFrame([features])


# Подготовка датасета
def prepare_dataset(json_data):
    """Перетворює масив JSON-об'єктів у навчальний датасет"""

    X = pd.DataFrame()
    y = []

    for item in json_data:
        features = extract_features(item['list'])
        X = pd.concat([X, features], ignore_index=True)
        y.append(item['mode'])

    # Заповнення пропусків (якщо часові ознаки відсутні)
    X = X.fillna(0)

    return X, np.array(y)


# Навчання моделі
def train_xgboost_model(X_train, y_train):
    """Навчає класифікатор XGBoost"""
    model = XGBClassifier(
        n_estimators=100,
        max_depth=5,
        learning_rate=0.1,
        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42,
        eval_metric='logloss'
    )
    model.fit(X_train, y_train)
    return model


# Приклад використання
if __name__ == "__main__":
    # Приклад вхідних даних
    sample_data = load_device_logs(1000)

    # Підготовка даних
    X, y = prepare_dataset(sample_data)
    print("Витягнуті ознаки:\n", X.head())

    # Розбиття на навчальну й тестову вибірки
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42)
```

```
# Вимірювання використання пам'яті до навчання
tracemalloc.start()
start_train = time.time()

# Навчання моделі
model = train_xgboost_model(X_train, y_train)
end_train = time.time()
training_time = end_train - start_train

# Вимірювання пам'яті після навчання
current, peak = tracemalloc.get_traced_memory()
max_ram_usage = peak / (1024 ** 2)  # у MB
tracemalloc.stop()

# Оцінка якості моделі
start_inf = time.time()
y_pred = model.predict(X_test)
end_inf = time.time()
inference_time = end_inf - start_inf

print(classification_report(y_test, y_pred))
print(f"Max RAM Usage: {max_ram_usage:.2f} MB")
print(f"Inference time: {inference_time:.4f} s")

```

Графові нейронні мережі (GNN – Graph Neural Networks) демонструють високу ефективність у задачах, де дані мають структурні залежності. У цьому випадку, якщо між елементами списку list існують неявні зв'язки — такі як часові закономірності (dateTime) або семантичні взаємовпливи (buttonKey) — GNN здатні виявляти ці взаємозв'язки завдяки своїй архітектурі, орієнтованій на обробку графів

Наприклад, якщо класифікація mode залежить від послідовності взаємодій або часових інтервалів між подіями, GNN можуть змоделювати такі залежності через агрегацію інформації від сусідніх вузлів графа. Крім того, GNN автоматично витягують значущі ознаки, усуваючи потребу в ручному конструюванні фіч, що особливо корисно при наявності складних нелінійних залежностей. Сучасні архітектури, такі як GraphSAGE або GAT (Graph Attention Networks), забезпечують масштабованість, дозволяючи обробляти списки змінної довжини та адаптуватися до динамічних даних

Однак застосування GNN може бути надмірним, якщо елементи в list статистично незалежні або якщо цільова змінна mode визначається простими агрегованими показниками, такими як частота появи певних buttonKey. У такому

випадку простіші методи — логістична регресія або ансамблі дерев рішень — можуть показати порівнянну точність при менших обчислювальних витратах

Ще одним обмеженням є вимогливість GNN до обсягу навчальних даних. При недостатній кількості розмічених прикладів виникає ризик перенавчання або нестабільності моделі. Крім того, інтерпретованість рішень, які приймає GNN, залишається низькою порівняно з традиційними ML-моделями, що може бути критично у сценаріях, де потрібна пояснюваність. Складність також становить коректне визначення топології графа. Якщо зв’язки між елементами списку неочевидні або відсутні, продуктивність GNN може не перевищувати базові методи

Використання GNN виправдане лише за наявності обґрунтованої гіпотези про графову природу даних. Таких обґрунтувань у нашому випадку немає, окрім порядку натискань кнопок пристроїв введення, незалежно від точної дати та часу

Нижче наведено приклад реалізації рішення з використанням GNN (Лістинг 6). Опис реалізації

Кожна подія в списку стає вузлом графа з трьома ознаками. Ознака buttonKey — це ідентифікатор кнопки, нормалізований. Ознака delta_time — це час між поточною та попередньою подією в секундах. Ознака normalized_ts — це час від початку послідовності, нормалізований

Опис ознак вузлів. buttonKey — пряма інформація про дію користувача. delta_time — потрібна, коли часові патерни можуть бути важливими, наприклад, при швидких і повільних послідовностях натискань. normalized_ts — враховує абсолютний час усередині послідовності

Опис топології графа. Використовується повнозв’язний граф, оскільки це дозволяє моделі виявити будь-які залежності між подіями, хоча це обчислювально затратне рішення для довгих послідовностей. Якщо mode залежить від глобальних патернів, наприклад, комбінацій кнопок — що цілком ймовірно — тоді повнозв’язний граф є кращим, тому обрана така топологія

Опис архітектури графа. Обрана архітектура GCN, оскільки це базова і добре вивчена архітектура, яка добре працює навіть на невеликих графах.

Використовується 2 шари GCN, оскільки 1 шар може призвести до недонавчання, а 3 і більше — до перенавчання. Операція global_mean_pool об’єднує інформацію з усіх вузлів. Для класифікації використовується бінарна крос-ентропія BCELoss. Функція вихідного сигналу — сигмоїда для отримання ймовірності класу 1, а не softmax, оскільки остання використовується для багатокласової класифікації, коли класів більше ніж два

Лістинг 7 - Фрагмент коду моделі GCN для обробки послідовностей подій.

```
import time
import tracemalloc

import torch
import torch.nn as nn
import torch.nn.functional as F
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from tensorflow.python.data.util.nest import flatten
from torch_geometric.data import Data, DataLoader
from torch_geometric.nn import GCNConv, global_mean_pool

from device_input.device_log_loader import load_device_logs


class EventGNN(nn.Module):
    def __init__(self, hidden_channels=64):
        super(EventGNN, self).__init__()
        # Враховуємо 3 ознаки на вузол: buttonKey, часові дельти, нормалізований timestamp
        self.node_encoder = nn.Linear(3, hidden_channels)

        # Два шари GCN з активацією ReLU
        self.conv1 = GCNConv(hidden_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)

        # Класифікатор
        self.classifier = nn.Sequential(
            nn.Linear(hidden_channels, hidden_channels),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(hidden_channels, 1)
        )

    def forward(self, x, edge_index, batch):
        # Прохід через GNN
        x = self.node_encoder(x)
        x = F.relu(self.conv1(x, edge_index))
        x = F.relu(self.conv2(x, edge_index))

        # Глобальний пулінг для отримання представлення всього графа
        x = global_mean_pool(x, batch)

        # Класифікація
        return torch.sigmoid(self.classifier(x)).squeeze(1)


def prepare_graph_data(sequences, labels):
    data_list = []

    for seq, label in zip(sequences, labels):
        # Витягуємо ознаки
        button_keys = torch.tensor([e['buttonKey'] for e in seq], dtype=torch.float)
        timestamps = [e['dateTime'] for e in seq]
        deltas = torch.tensor([
            (timestamps[i] - timestamps[i - 1]) if i > 0 else 0
```

```
            for i in range(len(seq))
        ], dtype=torch.float)
        normalized_ts = torch.tensor(
            [(ts - timestamps[0]) for ts in timestamps],
            dtype=torch.float
        )

        # Формуємо node features
        x = torch.stack([button_keys, deltas, normalized_ts], dim=1)

        # Будуємо повнозв'язний граф (можна замінити топологію)
        num_nodes = len(seq)
        edge_index = torch.tensor(
            [[i, j] for i in range(num_nodes)
             for j in range(num_nodes) if i != j],
            dtype=torch.long
        ).t()

        # Створюємо об'єкт Data
        data = Data(x=x, edge_index=edge_index, y=torch.tensor([label], dtype=torch.float))
        data_list.append(data)

    return data_list


# Приклад використання
if __name__ == "__main__":
    sample_data = load_device_logs(10)

    # Підготовка даних
    X = [item['list'] for item in sample_data]
    y = [item['mode'] for item in sample_data]
    graph_data = prepare_graph_data(X, y)

    # Розбиття на train/test
    train_data, test_data = train_test_split(graph_data, test_size=0.2, random_state=42)
    train_loader = DataLoader(train_data, batch_size=2, shuffle=True)
    test_loader = DataLoader(test_data, batch_size=2)

    # Ініціалізація моделі
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = EventGNN().to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
    criterion = nn.BCELoss()

    # Вимірювання використання пам'яті до навчання
    tracemalloc.start()
    start_train = time.time()

    # Навчання
    for epoch in range(10):
        model.train()
        total_loss = 0
```

```
109
110         for data in train_loader:
111             data = data.to(device)
112             optimizer.zero_grad()
113             out = model(data.x, data.edge_index, data.batch)
114             loss = criterion(out, data.y)
115             loss.backward()
116             optimizer.step()
117             total_loss += loss.item()
118
119         # Валідація
120         model.eval()
121         correct = 0
122         for data in test_loader:
123             data = data.to(device)
124             pred = (model(data.x, data.edge_index, data.batch) > 0.5).float()
125             correct += (pred == data.y).sum().item()
126
127         acc = correct / len(test_data)
128         print(f'Epoch {epoch}, Loss: {total_loss:.4f}, Test Acc: {acc:.4f}')
129
130     end_train = time.time()
131     training_time = end_train - start_train
132
133     # Вимірювання пам'яті після навчання
134     current, peak = tracemalloc.get_traced_memory()
135     max_ram_usage = peak / (1024 ** 2)  # у MB
136     tracemalloc.stop()
137
138     y_pred = []
139     inference_time = 0
140
141     with torch.no_grad():
142         start_inf = time.time()
143         for data in test_loader:
144             data = data.to(device)
145             pred = (model(data.x, data.edge_index, data.batch) > 0.5).float()
146             y_pred.extend(pred.cpu().numpy())
147         end_inf = time.time()
148         inference_time = end_inf - start_inf
149
150     # Формування тестових міток
151     y_test = (list(data.y) for data in test_loader)
152     y_test = [x for xs in y_test for x in xs]
153
154     print(classification_report(y_test, y_pred))
155     print(f"Max RAM Usage: {max_ram_usage:.2f} MB")
156     print(f"Inference time: {inference_time:.4f} s")
```

Свёрточна нейронна мережа (CNN – Convolutional Neural Network) демонструє високу ефективність у задачах автоматичного розпізнавання патернів, що може бути корисним для класифікації mode на основі послідовностей buttonKey і dateTime

Їх застосування має бути обґрунтоване структурою даних і характером залежностей. CNN здатні автоматично витягувати просторові та часові ознаки зі

структурованих даних без явного feature engineering. Якщо залежність між mode і послідовністю list визначається комбінаційними або локальними часовими патернами (наприклад, певні поєднання buttonKey або характерні інтервали між подіями), згорткові шари можуть ефективно їх виявляти. Крім того, CNN стійкі до зашумлених даних завдяки механізму фільтрації ознак через ядра згортки

При відповідній попередній обробці дані можуть бути перетворені у тензорну форму, наприклад, у вигляді матриці, де рядки відповідають подіям, а стовпці — ознакам (buttonKey, часові дельти тощо). У такому випадку 1D-згортки здатні аналізувати послідовності, а 2D-архітектура — виявляти складніші просторово-часові кореляції

Однак CNN не завжди є оптимальним рішенням. Якщо класифікація mode залежить від простих правил (наприклад, наявності конкретного buttonKey у list), надмірна параметризація CNN призведе до перенавчання, і більш ефективними виявляться класичні моделі, описані раніше. Ще одним обмеженням є обробка даних змінної довжини. На відміну від рекурентних архітектур (LSTM, GRU), CNN потребують фіксованого розміру входу, що змушує застосовувати паддінг або сегментацію, що може спотворити початкові часові залежності. Це вносить обмеження у вигляді відрізків, в межах яких відбувається класифікація, що не є критичним для нашої роботи

Крім того, CNN гірше справляються з довгостроковими часовими зв’язками, оскільки їх рецептивне поле обмежене розміром ядра згортки. Попередній аналіз даних показує наявність складних нелінійних патернів, тому CNN може бути виправданим, особливо при великому обсязі даних. Однак залежність між list і mode може визначатися і глобальними часовими тенденціями, тому може бути доцільніше використання рекурентних мереж

Нижче наведено приклад реалізації рішення з використанням CNN (Лістинг 7). Опис рішення

Попередня обробка. Послідовності мають різну довжину, фіксована довжина потрібна для роботи CNN у Keras, тому виконується доповнення нулями (padding) до фіксованої довжини max_sequence_length. Нульове доповнення

обране як найпростіший метод. Часові мітки мають строковий формат, тому виконується перетворення у datetime і обчислення часових дельт між подіями

Архітектура мережі. Використано 1D-згортки, а не 2D чи 3D, оскільки дані є часовими послідовностями, і кожен часовий крок містить 2 ознаки — номер кнопки і часова дельта. Параметри згорткових шарів підібрані так, що збільшення кількості фільтрів (спочатку 64, потім 128, потім 256) дозволяє виявляти спочатку прості, потім складні патерни. Параметр kernel_size = 3, оскільки це оптимально для аналізу локальних послідовностей. padding = 'same' — зберігає довжину послідовності

Застосовується BatchNormalization, оскільки це стабілізує навчання і дозволяє використовувати вищу частоту навчання. Застосовується GlobalAveragePooling1D замість Flatten, оскільки це зменшує кількість параметрів і має кращу здатність до узагальнення. Застосовується Dropout 0.5 у першому Dense-шарі як стандартне значення, 0.3 у другому — для балансу між регуляризацією і збереженням інформації

Підхід до навчання. Застосовується оптимізатор Adam, оскільки він забезпечує автоматичну адаптацію частоти навчання і добре працює за замовчуванням. Обрана функція втрат binary_crossentropy, оскільки це стандартний вибір для бінарної класифікації. Альтернативою є focal_loss для незбалансованих даних, однак ми можемо збалансувати дані іншими способами. Для підбору гіперпараметрів використовуються метрики Accuracy, оскільки це інтуїтивно зрозуміла метрика, і AUC, оскільки вона краще відображає якість на незбалансованих даних. Обрано 20 епох — це розумний компроміс між часом навчання і якістю. Batch size = 32, оскільки це стандартне значення для середніх наборів даних

Лістинг 8 — Рішення з використанням CNN

```python
import time
import tracemalloc

import torch
import torch.nn as nn
import torch.nn.functional as F
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from tensorflow.python.data.util.nest import flatten
from torch_geometric.data import Data, DataLoader
from torch_geometric.nn import GCNConv, global_mean_pool

from device_input.device_log_loader import load_device_logs


class EventGNN(nn.Module):
    def __init__(self, hidden_channels=64):
        super(EventGNN, self).__init__()
        # Враховуємо 3 ознаки на вузол: buttonKey, часові дельти, нормалізований timestamp
        self.node_encoder = nn.Linear(3, hidden_channels)

        # Два шари GCN з активацією ReLU
        self.conv1 = GCNConv(hidden_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)

        # Класифікатор
        self.classifier = nn.Sequential(
            nn.Linear(hidden_channels, hidden_channels),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(hidden_channels, 1)
        )

    def forward(self, x, edge_index, batch):
        # Прохід через GNN
        x = self.node_encoder(x)
        x = F.relu(self.conv1(x, edge_index))
        x = F.relu(self.conv2(x, edge_index))

        # Глобальний пулінг для отримання представлення всього графа
        x = global_mean_pool(x, batch)

        # Класифікація
        return torch.sigmoid(self.classifier(x)).squeeze(1)


def prepare_graph_data(sequences, labels):
    data_list = []

    for seq, label in zip(sequences, labels):
        # Витягуємо ознаки
        button_keys = torch.tensor([e['buttonKey'] for e in seq], dtype=torch.float)
        timestamps = [e['dateTime'] for e in seq]
        deltas = torch.tensor([
            (timestamps[i] - timestamps[i - 1]) if i > 0 else 0
```

```
            for i in range(len(seq))
        ], dtype=torch.float)
        normalized_ts = torch.tensor(
            [(ts - timestamps[0]) for ts in timestamps],
            dtype=torch.float
        )

        # Формуємо node features
        x = torch.stack([button_keys, deltas, normalized_ts], dim=1)

        # Будуємо повнозв'язний граф (можна замінити топологію)
        num_nodes = len(seq)
        edge_index = torch.tensor(
            [[i, j] for i in range(num_nodes)
             for j in range(num_nodes) if i != j],
            dtype=torch.long
        ).t()

        # Створюємо об'єкт Data
        data = Data(x=x, edge_index=edge_index, y=torch.tensor([label], dtype=torch.float))
        data_list.append(data)

    return data_list


# Приклад використання
if __name__ == "__main__":
    sample_data = load_device_logs(10)

    # Підготовка даних
    X = [item['list'] for item in sample_data]
    y = [item['mode'] for item in sample_data]
    graph_data = prepare_graph_data(X, y)

    # Розбиття на train/test
    train_data, test_data = train_test_split(graph_data, test_size=0.2, random_state=42)
    train_loader = DataLoader(train_data, batch_size=2, shuffle=True)
    test_loader = DataLoader(test_data, batch_size=2)

    # Ініціалізація моделі
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = EventGNN().to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
    criterion = nn.BCELoss()

    # Вимірювання використання пам'яті до навчання
    tracemalloc.start()
    start_train = time.time()

    # Навчання
    for epoch in range(10):
        model.train()
        total loss = 0
```

```
109
110         for data in train_loader:
111             data = data.to(device)
112             optimizer.zero_grad()
113             out = model(data.x, data.edge_index, data.batch)
114             loss = criterion(out, data.y)
115             loss.backward()
116             optimizer.step()
117             total_loss += loss.item()
118
119         # Валідація
120         model.eval()
121         correct = 0
122         for data in test_loader:
123             data = data.to(device)
124             pred = (model(data.x, data.edge_index, data.batch) > 0.5).float()
125             correct += (pred == data.y).sum().item()
126
127         acc = correct / len(test_data)
128         print(f'Epoch {epoch}, Loss: {total_loss:.4f}, Test Acc: {acc:.4f}')
129
130     end_train = time.time()
131     training_time = end_train - start_train
132
133     # Вимірювання пам'яті після навчання
134     current, peak = tracemalloc.get_traced_memory()
135     max_ram_usage = peak / (1024 ** 2)  # у MB
136     tracemalloc.stop()
137
138     y_pred = []
139     inference_time = 0
140
141     with torch.no_grad():
142         start_inf = time.time()
143         for data in test_loader:
144             data = data.to(device)
145             pred = (model(data.x, data.edge_index, data.batch) > 0.5).float()
146             y_pred.extend(pred.cpu().numpy())
147         end_inf = time.time()
148         inference_time = end_inf - start_inf
149
150     # Формування тестових міток
151     y_test = (list(data.y) for data in test_loader)
152     y_test = [x for xs in y_test for x in xs]
153
154     print(classification_report(y_test, y_pred))
155     print(f"Max RAM Usage: {max_ram_usage:.2f} MB")
156     print(f"Inference time: {inference_time:.4f} s")
```

Свёрточна нейронна мережа (CNN – Convolutional Neural Network) демонструє високу ефективність у задачах автоматичного розпізнавання патернів, що може бути корисним для класифікації mode на основі послідовностей buttonKey і dateTime

Їх застосування має бути обґрунтоване структурою даних і характером залежностей. CNN здатні автоматично витягувати просторові та часові ознаки зі

структурованих даних без явного feature engineering. Якщо залежність між mode і послідовністю list визначається комбінаційними або локальними часовими патернами (наприклад, певні поєднання buttonKey або характерні інтервали між подіями), згорткові шари можуть ефективно їх виявляти. Крім того, CNN стійкі до зашумлених даних завдяки механізму фільтрації ознак через ядра згортки

При відповідній попередній обробці дані можуть бути перетворені у тензорну форму, наприклад, у вигляді матриці, де рядки відповідають подіям, а стовпці — ознакам (buttonKey, часові дельти тощо). У такому випадку 1D-згортки здатні аналізувати послідовності, а 2D-архітектура — виявляти складніші просторово-часові кореляції

Однак CNN не завжди є оптимальним рішенням. Якщо класифікація mode залежить від простих правил (наприклад, наявності конкретного buttonKey у list), надмірна параметризація CNN призведе до перенавчання, і більш ефективними виявляться класичні моделі, описані раніше. Ще одним обмеженням є обробка даних змінної довжини. На відміну від рекурентних архітектур (LSTM, GRU), CNN потребують фіксованого розміру входу, що змушує застосовувати паддінг або сегментацію, що може спотворити початкові часові залежності. Це вносить обмеження у вигляді відрізків, в межах яких відбувається класифікація, що не є критичним для нашої роботи

Крім того, CNN гірше справляються з довгостроковими часовими зв’язками, оскільки їх рецептивне поле обмежене розміром ядра згортки. Попередній аналіз даних показує наявність складних нелінійних патернів, тому CNN може бути виправданим, особливо при великому обсязі даних. Однак залежність між list і mode може визначатися і глобальними часовими тенденціями, тому може бути доцільніше використання рекурентних мереж

Нижче наведено приклад реалізації рішення з використанням CNN (Лістинг 7). Опис рішення

Попередня обробка. Послідовності мають різну довжину, фіксована довжина потрібна для роботи CNN у Keras, тому виконується доповнення нулями (padding) до фіксованої довжини max_sequence_length. Нульове доповнення

обране як найпростіший метод. Часові мітки мають строковий формат, тому виконується перетворення у datetime і обчислення часових дельт між подіями

Архітектура мережі. Використано 1D-згортки, а не 2D чи 3D, оскільки дані є часовими послідовностями, і кожен часовий крок містить 2 ознаки — номер кнопки і часова дельта. Параметри згорткових шарів підібрані так, що збільшення кількості фільтрів (спочатку 64, потім 128, потім 256) дозволяє виявляти спочатку прості, потім складні патерни. Параметр kernel_size = 3, оскільки це оптимально для аналізу локальних послідовностей. padding = 'same' — зберігає довжину послідовності

Застосовується BatchNormalization, оскільки це стабілізує навчання і дозволяє використовувати вищу частоту навчання. Застосовується GlobalAveragePooling1D замість Flatten, оскільки це зменшує кількість параметрів і має кращу здатність до узагальнення. Застосовується Dropout 0.5 у першому Dense-шарі як стандартне значення, 0.3 у другому — для балансу між регуляризацією і збереженням інформації

Підхід до навчання. Застосовується оптимізатор Adam, оскільки він забезпечує автоматичну адаптацію частоти навчання і добре працює за замовчуванням. Обрана функція втрат binary_crossentropy, оскільки це стандартний вибір для бінарної класифікації. Альтернативою є focal_loss для незбалансованих даних, однак ми можемо збалансувати дані іншими способами. Для підбору гіперпараметрів використовуються метрики Accuracy, оскільки це інтуїтивно зрозуміла метрика, і AUC, оскільки вона краще відображає якість на незбалансованих даних. Обрано 20 епох — це розумний компроміс між часом навчання і якістю. Batch size = 32, оскільки це стандартне значення для середніх наборів даних

Лістинг 9 — Рішення з використанням CNN

```
import time
import tracemalloc

import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras import layers, models

from device_input.device_log_loader import load_device_logs


# 1. Попередня обробка даних
def preprocess_data(data, max_sequence_length=100):
    """
    Перетворює сирі дані у формат, придатний для навчання CNN.

    Args:
        data: Список словників у форматі {'mode': int, 'list': [{'buttonKey': int, 'dateTime': str}]}
        max_sequence_length: Максимальна довжина послідовності (доповнюється нулями)

    Returns:
        X: Тензор форми (n_samples, max_sequence_length, n_features)
        y: Масив міток
    """
    X = []
    y = []

    for item in data:
        # Отримуємо мітку
        y.append(item['mode'])

        sequence = item['list']
        features = []

        # Обробляємо кожну подію в послідовності
        for i, event in enumerate(sequence):
            if i >= max_sequence_length:
                break

            # Отримуємо ознаки події
            button_key = event['buttonKey']
            timestamp = pd.to_datetime(event['dateTime'])

            # Обчислюємо різницю часу з попередньою подією
            if i > 0:
                prev_time = pd.to_datetime(sequence[i - 1]['dateTime'])
                time_diff = (timestamp - prev_time).total_seconds()
            else:
                time_diff = 0.0

            features.append([button_key, time_diff])

```

```
            # Доповнюємо нулями, якщо послідовність коротша за max_sequence_length
            while len(features) < max_sequence_length:
                features.append([0, 0.0])

            X.append(features)

    return np.array(X), np.array(y)


# 2. Створення моделі CNN
def create_cnn_model(input_shape, num_classes=1):
    """
    Створює CNN модель для обробки часових послідовностей.

    Args:
        input_shape: Форма вхідних даних (max_sequence_length, n_features)
        num_classes: Кількість класів (1 для бінарної класифікації)

    Returns:
        Модель Keras
    """
    model = models.Sequential([
        # Нормалізація вхідних даних
        layers.BatchNormalization(input_shape=input_shape),

        # Перший згортковий блок
        layers.Conv1D(64, kernel_size=3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling1D(pool_size=2),

        # Другий згортковий блок
        layers.Conv1D(128, kernel_size=3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling1D(pool_size=2),

        # Третій згортковий блок
        layers.Conv1D(256, kernel_size=3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.GlobalAveragePooling1D(),

        # Повнозв'язні шари
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.3),

        # Вихідний шар
        layers.Dense(num_classes, activation='sigmoid')
    ])

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

```
    )

    return model


# 3. Навчання моделі
def train_model(X, y, epochs=20, batch_size=32):
    """
    Навчає модель CNN на наданих даних.

    Args:
        X: Вхідні дані
        y: Мітки
        epochs: Кількість епох
        batch_size: Розмір батчу

    Returns:
        Модель та історія навчання
    """

    # Створюємо модель
    model = create_cnn_model(input_shape=X.shape[1:])

    # Навчання
    history = model.fit(
        X, y,
        validation_data=(X_test, y_test),
        epochs=epochs,
        batch_size=batch_size,
        verbose=1
    )

    return model, history


# Приклад використання
if __name__ == "__main__":
    sample_data = load_device_logs(1000)

    # Попередня обробка
    X, y = preprocess_data(sample_data, max_sequence_length=50)

    # Поділ на train/test
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    # Нормалізація
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train.reshape(-1, X_train.shape[-1])).reshape(X_train.shape)
    X_test = scaler.transform(X_test.reshape(-1, X_test.shape[-1])).reshape(X_test.shape)

    # Вимір використання пам'яті
    tracemalloc.start()
```

```
# Вимір використання пам'яті
tracemalloc.start()
start_train = time.time()

# Навчання моделі
model, history = train_model(X_train, y_train, epochs=10)

end_train = time.time()
training_time = end_train - start_train

# Використання пам'яті після навчання
current, peak = tracemalloc.get_traced_memory()
max_ram_usage = peak / (1024 ** 2)  # MB
tracemalloc.stop()

# Збереження моделі
model.save("button_sequence_classifier.h5")

# Оцінка якості
start_inf = time.time()
y_pred = model.predict(X_test)
end_inf = time.time()

inference_time = end_inf - start_inf
y_pred = [round(num) for sublist in y_pred for num in sublist]

print(classification_report(y_test, y_pred))
print(f"Max RAM Usage: {max_ram_usage:.2f} MB")
print(f"Inference time: {inference_time:.4f} s")

```

LSTM (Long Short-Term Memory) демонструє високу ефективність при обробці послідовностей даних, таких як часові ряди або логування подій, завдяки своїй архітектурі, здатній вловлювати довгострокові залежності

У контексті задачі класифікації mode на основі послідовності buttonKey і відповідних часових міток dateTime, ключовою перевагою LSTM є здатність автоматично виявляти складні патерни без потреби в ручному конструюванні ознак. Це особливо важливо, якщо залежність між послідовністю подій і цільовою змінною є нелінійною або включає приховані часові кореляції

Крім того, LSTM стійка до проблеми зникання градієнтів, що дозволяє ефективно навчатися на довгих послідовностях. Модель здатна обробляти вхідні дані змінної довжини, що робить її гнучкою в умовах нефіксованого розміру list. За наявності достатнього обсягу розмічених даних LSTM може досягати високої точності, перевершуючи традиційні алгоритми машинного навчання, особливо в задачах, де часова динаміка відіграє ключову роль

Однак застосування LSTM має низку обмежень. Основне — висока обчислювальна складність навчання, що потребує значних ресурсів, включаючи використання GPU або TPU для прискорення процесу. Це робить LSTM менш придатною в умовах обмежених обчислювальних потужностей або потреби в швидкому інференсі

Іншим суттєвим недоліком є вимога до великого обсягу навчальних даних. При недостатній кількості прикладів модель схильна до перенавчання, що призводить до поганої здатності до узагальнення. Крім того, інтерпретованість LSTM залишається низькою: внутрішні механізми роботи моделі є «чорним ящиком», що ускладнює аналіз прийняття рішень і робить її менш придатною в галузях, де потрібна пояснюваність

Нижче наведено приклад реалізації рішення з використанням RNN-LSTM (Лістинг 8). Опис рішення

Як і в попередньому рішенні з використанням нейронних мереж, ми масштабуємо дані та приводимо їх до необхідного вигляду — використовуються MinMaxScaler, pad_sequences

Опис шарів мережі. Вхідний Masking ігнорує нульові доповнення (mask_value = 0), щоб модель не обробляла «порожні» події, додані при паддінгу. Без Masking мережа навчалася б на нульових значеннях, що знижує якість класифікації

Далі йде один LSTM-шар на 64 вузли, де return_sequences = False — він повертає лише останній вихід послідовності, оскільки завданням є бінарна класифікація всієї послідовності (mode = 0 або 1), а не прогноз для кожного кроку. Розмір у 64 нейрони обрано як компроміс між швидкістю та здатністю вловлювати складні патерни

Застосовується Dropout (0.2) — регуляризація для запобігання перенавчанню, значення 0.2 обрано емпірично. Вихідний шар — Dense з активацією sigmoid, обраною з тієї ж причини, що і в рішенні з нейронною мережею вище

Оптимізатор, функція втрат, метрики, розмір батчу, кількість епох — підібрані за тими ж міркуваннями, що й раніше

Лістинг 8 — Приклад реалізації рішення з використанням RNN-LSTM

```
import time
import tracemalloc

import numpy as np
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.layers import LSTM, Dense, Masking, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.sequence import pad_sequences

from device_input.device_log_loader import load_device_logs


def prepare_dataset(json_data):
    """Готує датасет із сирих JSON-даних"""
    X = []
    y = []

    for item in json_data:
        # Отримуємо цільову мітку
        y.append(item['mode'])

        # Отримуємо події
        X.append(item['list'])

    return X, y


# Перетворення даних у пітру-масиви
def prepare_data(X, y):
    # Витягуємо buttonKey та dateTime з кожної послідовності
    X_processed = []
    for seq in X:
        seq_data = []
        for event in seq:
            seq_data.append([event['buttonKey'], event['dateTime']])
        X_processed.append(seq_data)

    # Перетворюємо у пітру та нормалізуємо
    X_processed = np.array(X_processed, dtype='float32')
    scaler = MinMaxScaler()

    # Об'єднуємо всі події для нормалізації
    X_reshaped = X_processed.reshape(-1, 2)
    X_scaled = scaler.fit_transform(X_reshaped)
    X_scaled = X_scaled.reshape(X_processed.shape)  # Повертаємо форму назад

    # Додаємо паддінг, якщо послідовності різної довжини
    max_len = max(len(seq) for seq in X)
    X_padded = pad_sequences(X_scaled, maxlen=max_len, padding='post', dtype='float32')

    y = np.array(y, dtype='float32')
    return X_padded, y
```

```python
# Приклад використання
if __name__ == "__main__":
    sample_data = load_device_logs(1000)
    X, y = prepare_dataset(sample_data)

    X_prepared, y_prepared = prepare_data(X, y)

    # Розділення на train/test
    X_train, X_test, y_train, y_test = train_test_split(
        X_prepared, y_prepared, test_size=0.2, random_state=42
    )

    # Створення LSTM-моделі
    model = Sequential([
        Masking(mask_value=0., input_shape=(X_train.shape[1], X_train.shape[2])),  # Ігнорує нульове доповнення
        LSTM(64, activation='tanh', return_sequences=False),
        Dropout(0.2),
        Dense(1, activation='sigmoid')  # Бінарна класифікація
    ])

    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='binary_crossentropy',
        metrics=['accuracy']
    )

    # Вимірювання використання пам'яті до навчання
    tracemalloc.start()
    start_train = time.time()

    # Навчання
    history = model.fit(
        X_train, y_train,
        validation_data=(X_test, y_test),
        epochs=20,
        batch_size=32
    )

    end_train = time.time()
    training_time = end_train - start_train

    # Вимірювання пам'яті після навчання
    current, peak = tracemalloc.get_traced_memory()
    max_ram_usage = peak / (1024 ** 2)  # у MB
    tracemalloc.stop()

    # Оцінка якості
    start_inf = time.time()
    y_pred = model.predict(X_test)
    end_inf = time.time()
    inference_time = end_inf - start_inf
```

```python
    y_pred = [round(num) for sublist in y_pred for num in sublist]

    print(classification_report(y_test, y_pred))
    print(f"Max RAM Usage: {max_ram_usage:.2f} MB")
    print(f"Inference time: {inference_time:.4f} s")
```

Gated Recurrent Unit (GRU) — ефективна архітектура рекурентних нейронних мереж (RNN) для обробки послідовних даних із часовими залежностями

У контексті задачі класифікації mode на основі часового ряду натискань кнопок (buttonKey) і відповідних часових міток (dateTime), GRU демонструє низку переваг. По-перше, завдяки наявності механізмів керованих воріт (update gate і reset gate), GRU здатна виявляти довгострокові залежності в даних, що

критично важливо для коректної класифікації, якщо цільова ознака mode залежить від порядку або часових інтервалів між подіями

По-друге, порівняно з Long Short-Term Memory (LSTM), GRU має меншу кількість параметрів, що знижує ризик перенавчання при обмеженому обсязі даних. По-третє, модель автоматично витягує значущі ознаки з сирих послідовностей, усуваючи потребу в ручному конструюванні фіч — таких як статистичні агрегати або часові дельти

Незважаючи на переваги, застосування GRU має низку обмежень. Основна проблема — висока чутливість до обсягу навчальних даних: при недостатній кількості прикладів модель схильна до перенавчання, особливо якщо часові залежності слабо виражені або нерелевантні для класифікації. Крім того, інтерпретованість GRU залишається низькою через її «чорну скриньку», що ускладнює аналіз значущості окремих часових кроків або ознак

З точки зору попередньої обробки даних, GRU потребує ретельної нормалізації часових міток (наприклад, перетворення в відносні інтервали) і коректного представлення категоріальних ознак (наприклад, використання embedding-шарів для buttonKey). Обчислювальна складність навчання також вища порівняно з традиційними методами машинного навчання, такими як ансамблі дерев, що може бути критично в умовах обмежених ресурсів

Нижче наведено приклад реалізації рішення з використанням RNN-GRU

Лістинг 10 — Реалізація рішення з використанням RNN-GRU

```
import time
import tracemalloc
from datetime import datetime

import numpy as np
import tensorflow as tf
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import (GRU, Dense, Dropout, Embedding,
                                     Masking, Lambda, Concatenate, Reshape, Input)
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.sequence import pad_sequences

from device_input.device_log_loader import load_device_logs


def prepare_dataset(json_data):
    """Готує датасет із сирих JSON-даних"""
    X = []
    y = []

    for item in json_data:
        # Отримуємо цільову мітку
        y.append(item['mode'])

        # Отримуємо події
        X.append(item['list'])

    return X, y


class GRUClassifier:
    def __init__(self, max_sequence_length=100, num_button_keys=100):
        self.max_sequence_length = max_sequence_length
        self.num_button_keys = num_button_keys
        self.model = self._build_model()

    def _build_model(self):
        """Побудова архітектури моделі з коректною обробкою типів даних"""
        # Вхідний шар
        input_layer = Input(shape=(self.max_sequence_length, 2))

        # Маскування
        masked = Masking(mask_value=0.)(input_layer)

        # Розділення вхідних даних
        button_key = Lambda(lambda x: x[:, :, 0])(masked)
        time_delta = Lambda(lambda x: x[:, :, 1])(masked)

        # Перетворення buttonKey у цілочисельний тип
        button_key_int = Lambda(lambda x: tf.cast(x, 'int32'))(button_key)

        # Embedding-шар
        embedded = Embedding(
```

```
            input_dim=self.num_button_keys + 1,
            output_dim=16,
            input_length=self.max_sequence_length,
            mask_zero=True
        )(button_key_int)

        # Обробка часових ознак
        time_processed = Reshape((self.max_sequence_length, 1))(time_delta)

        # Об'єднання ознак
        concatenated = Concatenate(axis=-1)([embedded, time_processed])

        # GRU-шари
        gru1 = GRU(64, return_sequences=True, dropout=0.2, recurrent_dropout=0.2)(concatenated)
        gru2 = GRU(32, dropout=0.2, recurrent_dropout=0.2)(gru1)

        # Повнозв'язні шари
        dense1 = Dense(32, activation='relu')(gru2)
        dropout = Dropout(0.5)(dense1)
        output = Dense(1, activation='sigmoid')(dropout)

        model = Model(inputs=input_layer, outputs=output)

        model.compile(
            optimizer=Adam(learning_rate=0.001),
            loss='binary_crossentropy',
            metrics=['accuracy']
        )

        return model

    def _preprocess_data(self, X_raw, y_raw):
        """Попередня обробка даних з контролем типів"""
        processed_sequences = []

        for sequence in X_raw:
            # Обробка часових міток
            timestamps = [event['dateTime'] for event in sequence]
            if isinstance(timestamps[0], str):
                timestamps = [datetime.strptime(ts, '%Y-%m-%d %H:%M:%S').timestamp() for ts in timestamps]

            time_deltas = [0] + [timestamps[i] - timestamps[i - 1] for i in range(1, len(timestamps))]
            max_delta = max(time_deltas) if max(time_deltas) > 0 else 1
            normalized_deltas = [delta / max_delta for delta in time_deltas]

            # buttonKey → int, delta → float
            sequence_features = [
                [int(event['buttonKey']), float(normalized_deltas[i])]
                for i, event in enumerate(sequence)
            ]

            processed_sequences.append(sequence_features)

        # Паддинг
        X_padded = pad_sequences(
```

```
        X_padded = pad_sequences(
            processed_sequences,
            maxlen=self.max_sequence_length,
            dtype='float32',
            padding='post',
            truncating='post',
            value=0.0
        )

        y_array = np.array(y_raw, dtype='float32')

        return X_padded, y_array

    def train(self, X_train, y_train, X_val=None, y_val=None, epochs=20, batch_size=32):
        """Навчання моделі"""
        X_processed, y_processed = self._preprocess_data(X_train, y_train)

        if X_val is not None:
            X_val_processed, y_val_processed = self._preprocess_data(X_val, y_val)
            validation_data = (X_val_processed, y_val_processed)
        else:
            validation_data = None

        history = self.model.fit(
            X_processed, y_processed,
            validation_data=validation_data,
            epochs=epochs,
            batch_size=batch_size,
            verbose=1
        )
        return history

    def evaluate(self, X_test, y_test):
        """Оцінювання моделі на тестовій вибірці"""
        X_processed, y_processed = self._preprocess_data(X_test, y_test)
        loss, accuracy = self.model.evaluate(X_processed, y_processed, verbose=0)
        y_pred = (self.model.predict(X_processed) > 0.5).astype("int32")
        print(f"Test Accuracy: {accuracy:.4f}")
        print(classification_report(y_processed, y_pred))
        return y_pred

    def predict(self, X_new):
        """Передбачення класу для нових даних"""
        X_processed, _ = self._preprocess_data(X_new, [0] * len(X_new))
        return (self.model.predict(X_processed) > 0.5).astype("int32")


# Приклад використання
if __name__ == "__main__":
    sample_data = load_device_logs(1000)
    X, y = prepare_dataset(sample_data)

    # Розділення на train/test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
    # Вимірювання використання пам'яті до навчання
    tracemalloc.start()
    start_train = time.time()

    # Ініціалізація і навчання моделі
    classifier = GRUClassifier(max_sequence_length=50, num_button_keys=80)
    classifier.train(X_train, y_train, epochs=5)

    end_train = time.time()
    training_time = end_train - start_train

    # Вимірювання пам'яті після навчання
    current, peak = tracemalloc.get_traced_memory()
    max_ram_usage = peak / (1
```

3.2 Навчання моделей машинного навчання та інтеграція в систему

Проведене порівняльне дослідження алгоритмів машинного навчання для модуля оцінки активності користувача дозволило виявити суттєві відмінності в їх ефективності. Результати експериментальних випробувань демонструють, що найвищу точність класифікації з показником 0,90 забезпечують три методи: випадковий ліс, згорткові нейронні мережі та рекурентні мережі з механізмом довгої короткострокової пам'яті. Помірну ефективність із значенням точності 0,70 показала логістична регресія, тоді як графові нейронні мережі та статистичний підхід з пороговими правилами не перевищили рівня випадкового вгадування з показником 0,50. Найменш ефективними виявилися мережі GRU із точністю лише 0,30.

Детальний аналіз метрик якості підтверджує лідерство випадкового лісу, CNN та LSTM. Для класу непродуктивної активності значення F1-міри складає 0,89-0,93, а для класу продуктивної активності – 0,80-0,86. Критично низьку ефективність демонструють GNN, які повністю не змогли класифікувати об'єкти продуктивного класу, та GRU, що показали незадовільні результати щодо розпізнавання непродуктивної активності.

Важливим аспектом дослідження стала оцінка обчислювальної ефективності методів. Випадковий ліс виявився найбільш оптимальним рішенням з часом класифікації 0,0040 секунди та споживанням пам'яті 0,21 МБ. На противагу цьому, глибокі нейронні мережі вимагали значно більших ресурсів: об'єм оперативної пам'яті становив 14-24 МБ, а час інференсу сягав 2,5 секунди.

На підставі отриманих результатів сформульовано наступні рекомендації щодо вибору методу класифікації. Випадковий ліс є оптимальним рішенням для практичного впровадження, оскільки поєднує високу точність із мінімальними обчислювальними витратами. CNN та LSTM доцільно застосовувати в задачах, де пріоритетом є максимальна точність класифікації без урахування витрат ресурсів. Логістична регресія може слугувати компромісним варіантом за умови обмеженої обчислювальної потужності системи. GNN та GRU показали

незадовільні результати й не рекомендуються для використання в практичних реалізаціях.

Таким чином, для фінальної реалізації системи моніторингу обрано метод випадкового лісу, що забезпечує оптимальний баланс між точністю класифікації та ефективністю використання обчислювальних ресурсів, що є вирішальним фактором для систем реального часу.

Переходимо до розробки алгоритму модуля оцінки активності на основі переданих списків запущених процесів на комп'ютері

Нижче представлено модель для оцінки активності користувача комп'ютера, яка працює з такими даними: дата та час, список назв запущених процесів, мітка виду активності користувача для навчання моделі.

Листинг 11 – Код модели оценки по списку процессов

```
model = Sequential()
amount_of_different_words = (len(tokenizer.word_index) + 1)
model.add(
    Embedding(input_dim=amount_of_different_words, output_dim=amount_of_different_words, input_length=max_length))
model.add(LSTM(amount_of_different_words))
model.add(Dense(2, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Модель, представлена в коді, являє собою послідовну (Sequential) архітектуру штучної нейронної мережі, що включає шар Embedding, шар LSTM (Long Short-Term Memory) та повнозв'язний шар (Dense) з функцією активації softmax (Додаток 1).

Ця архітектура призначена для завдань класифікації текстів із двома вихідними класами. Її доцільно застосовувати в задачах бінарної класифікації, де важливими є контекст та порядок слів. В нашому випадку вхідні дані представлені у вигляді набору текстових назв процесів, що за своєю структурою аналогічно тексту.

Шар Embedding перетворює дискретні токени (назви процесів) у щільні вектори фіксованої розмірності. Це дозволяє моделі працювати з текстовими даними в числовому представленні. Далі йде **шар LSTM** - різновид рекурентної нейронної мережі, здатної запам'ятовувати довгострокові залежності в послідовностях. Це особливо важливо, оскільки порядок і контекст появи процесів може бути критичним для визначення активності користувача.

LSTM ефективно долає проблему зникаючого градієнта, що робить його переважним вибором для аналізу довгих послідовностей. Однак внутрішні механізми LSTM важко інтерпретувати, що може бути недоліком у задачах, де потрібна пояснюваність. В даному випадку це не є критичним.

Якість моделі значною мірою залежить від якості токенізації та попередньої обробки тексту. За наявності шуму чи нерелевантних слів можлива деградація якості, однак у цій задачі всі елементи списку є валідними, що знижує ризик спотворень. Використання функції активації **softmax** у вихідному шарі дозволяє інтерпретувати результат як ймовірності належності до кожного з двох класів, що зручно для оцінки впевненості моделі.

Оптимізатор Adam поєднує переваги адаптивних методів (наприклад, RMSprop) та методу моментів, що забезпечує швидку збіжність і стійкість до неоднорідності даних. Він також автоматично регулює швидкість навчання, спрощуючи налаштування моделі.

Модель використовує фіксовану довжину вхідних послідовностей (max_length). Це може призвести до втрати інформації при обрізанні довгих послідовностей або до надмірного паддингу при коротких. Для мінімізації втрат у попередній обробці застосовується сортування списку процесів за навантаженням процесора та іншими непрямими ознаками активності, що дозволяє зберегти найбільш значущі елементи.

Для роботи моделі використовується попередня токенізація назв процесів (Додаток 8). Результати токенізації зберігаються в окремий файл, що забезпечує повторне використання між сесіями навчання. При появі нових унікальних слів словник автоматично доповнюється, що підвищує стійкість моделі до мінливого складу процесів.

Лістинг 12 – Токенізація словника назв процесівdef process_tokenization(texts: [str], filename='tokens_dictionary.json'):

```
# Завантажуємо або створюємо токенізатор
tokenizer = load_or_create_tokenizer()

# Оновлюємо токенізатор новими текстами
update_tokenizer(tokenizer, texts, filename)

# Виводимо поточний словник токенів
# print("Словник токенів:", tokenizer.word_index)

return tokenizer
```

Також для збору даних використовуються скрипти для збору списків запущених процесів та збору даних з пристроїв вводу

```
def save_processes_to_file(dir_to_logs, is_working_mode_target_value):
    processes = []
    processes_seen = set()  # Використовуємо множину для відстеження вже побачених кортежів (name, pid).
    for p in psutil.process_iter(attrs=['name', 'pid', 'cpu_percent']):
        if hasattr(p, 'info'):
            try:
                # Отримуємо навантаження на CPU
                p.cpu_percent(interval=0)  # Для отримання актуального значення надалі
                if p.info['name'] not in processes_seen:
                    processes_seen.add(p.info['name'])  # Позначаємо процес як оброблений
                    processes.append((p.info['name'], p.info['pid'], p.cpu_percent(interval=0)))
            except (psutil.NoSuchProcess, psutil.AccessDenied):
                continue

    # Сортуємо процеси за навантаженням CPU та за PID у спадному порядку
    sorted_processes = sorted(processes, key=lambda x: (x[2], x[1]), reverse=True)
```

Лістинг 13 - Сбор данных о запущенных процессах

Далі розглянемо модель для роботи зі зібраними даними з пристроїв вводу, таких як клавіатура та миша (Error! Reference source not found.)

```
def log_event(button_key, is_working_mode):
    timestamp = datetime.now().isoformat()  # Беремо поточний час
    log_entry = {
        "timestamp": timestamp,
        "buttonKey": button_key,
        "isWorkingMode": is_working_mode
    }
    logs.append(log_entry)
    with open('device_logs.json', 'w', encoding='utf-8') as f:
        json.dump(logs, f, indent=4)
```

Лістинг 14 – Код моделі оцінки за даними з пристроїв вводу

```
model = RandomForestClassifier()
model.fit(X_train, y_train)
```

Наведений код демонструє процес підготовки даних та навчання моделі класифікації з використанням методу Random Forest (випадковий ліс) (Додаток 2). Ми не використовуємо додаткових інструментів моделі, оскільки дане налаштування є достатнім для ефективної оцінки активності користувача.

Розглянемо переваги та недоліки даного підходу, а також ситуації, коли його застосування є доцільним. У випадку роботи з різнорідними даними Random Forest може бути хорошим вибором, тому він був використаний. Необхідний аналіз важливості ознак, оскільки пристрої вводу різні, тип вводу також відрізняється, що потребує врахування даної обставини. Низька розмірність даних дозволяє уникнути підвищеного навантаження на обчислювальну систему. Важко інтерпретувати навчену модель, однак у нашому випадку це не є обов'язковою вимогою.

Основною частиною є функція, де запускаються навчені моделі та отримується їх усереднений результат [Лістинг 14, Додаток 3]. Навчені моделі завантажуються із збережених файлів, створених під час навчання моделей (Додаток 6, Додаток 7).

Лістинг 15 - Запуск навчених моделей та усереднення їх прогнозів

```
def main():
    print('predict_device_input.predict()')
    print(predict_device_input.predict())
    process_names_prediction = predict_process.predict()[0][0]
    device_input_prediction = calculate_true_proportion(predict_device_input.predict())
    ret = process_names_prediction / 2 + device_input_prediction / 2
    return ret
```

3.3. Тестування системи та аналіз результатів

Розроблена система моніторингу представляє собою комплексне програмне рішення для аналізу продуктивності працівників IT-компаній Система побудована на принципах багаторівневої обробки даних та автоматичного прийняття рішень що дозволяє ефективно класифікувати робочу активність та вчасно втручатися у випадку виявлення непродуктивної діяльності

Ключові компоненти системи

Основними учасниками системи є співробітник який безпосередньо використовує робочу станцію та менеджер що здійснює оперативний контроль за робочими процесами Співробітник взаємодіє з персональним комп'ютером виконуючи свої службові обов'язки а система в цей час здійснює безперервний

моніторинг його дій Менеджер отримує зведену інформацію про активність працівників і може втручатися у робочий процес при необхідності

Процес збору та аналізу даних

Система здійснює комплексний збір даних про взаємодію користувача з робочою станцією Відстежуються всі аспекти роботи від активності пристроїв введення клавіатури та миші до переліку запущених додатків і системних процесів Отримана інформація піддається багатоетапному аналізу з використанням передових алгоритмів машинного навчання що забезпечує точну класифікацію типу діяльності користувача

Механізми прийняття рішень

Особливістю системи є гнучкий механізм оцінки необхідності обмеження доступу до робочої станції Алгоритм враховує сукупність факторів включаючи результати паралельного аналізу від різних систем моніторингу статистичну значущість отриманих даних порогові значення ймовірностей для різних типів активності та контекст робочого процесу Система передбачає два основних сценарії роботи автоматичний режим при однозначних результатах аналізу та режим кваліфікованого прийняття рішень з участю менеджера при невизначеності в оцінках

Функціональні модулі системи

Архітектура системи інтегрує кілька взаємопов'язаних модулів Модуль збору телеметрії відповідає за фіксацію дій користувача в реальному часі Аналітичний блок обробляє отримані дані за допомогою передбачувальних моделей машинного навчання Система прийняття рішень ґрунтується на ймовірнісних оцінках та порівнянні з емпірично встановленими критеріями Механізм контролю доступу безпосередньо керує рівнем доступу до робочих ресурсів а звітний модуль забезпечує генерацію детальних звітів для візуалізації результатів моніторингу

Процедура обробки інформації

Процес обробки інформації включає складну послідовність операцій починаючи від первинного накопичення та класифікації даних до формування

агрегованих показників і порівняння з встановленими критеріями Система здійснює групування записів за часовими характеристиками синхронізацію пакетів даних та їх подальший аналіз Механізм прийняття рішень оцінює достовірність результатів аналізу та порівнює сумарні показники з пороговими значеннями для ініціації змін у рівні доступу

Адаптивність та масштабованість системи

Архітектура системи забезпечує широкі можливості для подальшого розширення функціоналу та адаптації під специфічні вимоги конкретних організацій Модульна структура дозволяє інтегрувати додаткові джерела даних та методи аналізу без необхідності фундаментальної перебудови існуючої інфраструктури Система здатна ефективно функціонувати в умовах зростання обсягів даних та ускладнення аналітичних завдань завдяки масштабованим алгоритмам обробки інформації

Генерація звітності та моніторинг

Система генерує комплексні звіти що дозволяють оцінити як поточну продуктивність працівників так і динаміку змін у їх роботі на протязі часу Звіти включають детальну інформацію про активність користувачів результати класифікації типу діяльності та статистику щодо прийнятих рішень про обмеження доступу Ця інформація є цінним джерелом для аналізу ефективності трудового процесу та оптимізації робочих місць

ВИСНОВКИ

У рамках даного дослідження було успішно розроблено автоматизовану систему моніторингу активності співробітників IT-компаній, що представляє собою комплексне програмне рішення для аналізу продуктивності праці та попередження техностресу. В ході роботи було вирішено низку ключових завдань, спрямованих на досягнення поставленої мети.

Проведено глибоке теоретико-методичне дослідження предметної області, що включало аналіз сучасних підходів до моніторингу активності персоналу, особливостей праці співробітників IT-сектору та проблем техностресу. Це дозволило визначити оптимальні шляхи вирішення поставлених завдань та сформувати наукову основу для розробки системи.

Було обґрунтовано вибір методів та інструментальних засобів управління IT-проектом, зокрема використання каскадної моделі Waterfall для основних етапів розробки та інструментів Kanban для оперативного управління завданнями. Застосовано GitHub для контролю версій та спільної роботи над кодом.

Розроблено детальне технічне завдання на реалізацію IT-проєкту, яке відповідає вимогам стандартів і регламентів розробки програмного забезпечення. Технічне завдання включає повний опис функціональних вимог, вимог до надійності, умов експлуатації та техніко-економічних показників системи.

Створено алгоритми для вирішення задач дослідження, зокрема реалізовано порівняльний аналіз різних методів машинного навчання для класифікації активності користувача. Доведено ефективність використання методу випадкового лісу (Random Forest) для аналізу даних з пристроїв вводу та LSTM-мережі для обробки переліків запущених процесів.

Розроблена система демонструє високу точність класифікації (до 90%) при мінімальному споживанні обчислювальних ресурсів, що робить її придатною для використання в реальних умовах роботи IT-компаній. Система successfully інтегрує різні джерела даних та забезпечує автоматичне прийняття

рішень щодо регулювання доступу до робочих станцій на основі аналізу активності користувачів.

Практична цінність роботи полягає в тому, що розроблена система може бути використана для оптимізації робочого часу, запобігання емоційному вигоранню співробітників та покращення загального рівня цифрового благополуччя в IT-колективах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zhang H. Personal information organization and re-access in computer folders: an empirical study of information workers 2011.
2. Sherif K., Jewesimi O. Electronic performance monitoring friend or foe? Empowering employees through data analytics 2018.
3. Patwardhan A., Knapp G. EmoFit: Affect Monitoring System for Sedentary Jobs 2016.
4. Nepal S. [et al.] From User Surveys to Telemetry-Driven Agents: Exploring the Potential of Personalized Productivity Solutions 2024.
5. Al Samman A. M., Althawaini F. The Impact of Electronic Monitoring on Employee Performance: Enhancing Transparency, Accountability, and Productivity 2024.
6. Arora S. Enhancing Employee Well-being with Machine Learning: Predictive Models for Health and Wellness Programs Journal of Informatics Education and Research 2024.
7. Azpíroz-Dorronsoro C. [et al.] Technostress and work-family conflict in ICT-user employees during the COVID-19 pandemic: the role of social support and mindfulness Behaviour & Information Technology 2024.
8. Baek I. G., Ko Y. H. Effects of an Electronic Monitoring System on Employees' Productivity in Telecommuting Arrangements Academy of Management Proceedings 2022.
9. Das A. K. [et al.] Enhancing Workplace Efficiency and Security Through Intelligent Employee Surveillance International Journal of Innovative Science and Research Technology (IJISRT) 2024.
10. Demerouti E. [et al.] The job demands-resources model of burnout Journal of Applied Psychology 2001.
11. Donati M. [et al.] RT-PROFASY: Enhancing the Well-being, Safety and Productivity of Workers by Exploiting Wearable Sensors and Artificial Intelligence 2022.
12. Dowrie S., Belle J.-P. V., Turpin M. Employee Technostress in South Africa's Hybrid Workplaces: Causes and Coping Mechanisms 2023.
13. Du P. Research on the training system of enterprise employees' digital skills enhancement Advances in Economics and Management Research 2023.
14. Dutta D., Mishra S. K. "Technology is killing me!": the moderating effect of organization home-work interface on the linkage between technostress and stress at work Information Technology & People 2023.
15. Fellmann M., Lambusch F., Schmidt A. C. Combining Computer-Based Activity Tracking with Human Energy and Sentiment Self-assessment for Continuous Work-Life Reflection Cham: Springer Nature Switzerland 2023.
16. Harunavamwe M., Kanengoni H. Hybrid and virtual work settings; the interaction between technostress, perceived organisational support, work-family conflict and the impact on work engagement African Journal of Economic and Management Studies 2023.

17. Khedhaouria A. [et al.] Consequences of technostress for users in remote (home) work contexts during a time of crisis: The buffering role of emotional social support Technological Forecasting and Social Change 2024.

18. Kolbeinsson Ö. [et al.] No sound is more distracting than the one you're trying not to hear: delayed costs of mental control of task-irrelevant neutral and emotional sounds BMC Psychology 2022.

19. Mordi C., Akanji B., Ajonbadi H. Exploring the impact of technostress on the work-life boundary of UK academics during the coronavirus pandemic Information Technology & People 2025.

20. Palczyńska M., Rynko M. ICT skills measurement in social surveys: Can we trust self-reports? Quality & Quantity 2021.

21. Pazio O. [et al.] Assessment of physical activity of people employed in the IT sector during the COVID-19 pandemic Folia Cardiologica 2022.

22. Reddy Katam B. AI-Driven Mood Analysis for Employee Wellbeing: A Proactive Approach to Enhancing Workplace Productivity International Journal of Innovative Science and Research Technology (IJISRT) 2024.

23. Schmidt B. Information Work Support based on Activity Data.

24. Sherif K., Jewesimi O., El-Masri M. Empowering employees: the other side of electronic performance monitoring Journal of Information, Communication and Ethics in Society 2020.

25. Tajitsu Y. [et al.] A Prototype Sensor System Using Fabricated Piezoelectric Braided Cord for Work-Environment Measurement during Work from Home Micromachines 2021.

26. Tariq Beigh F. [et al.] Intelligent Workplace Activity Monitoring and Detection Using Self-Powered Triboelectric/Piezoelectric Sensor Augmented Machine Learning IEEE Sensors Letters 2023.

27. Vinith M. K., Pinto M. D. E. INFORMATION TECHNOLOGY IN SURVEILLANCE OF EMPLOYEE PERFORMANCE EPRA International Journal of Environmental Economics, Commerce and Educational Management 2023.

28. Voigt J. [et al.] Conceptional Framework for the Objective Work-Related Quality of Life Measurement Through Multimodal Data Integration from Wearables and Digital Interaction 2024.

29. Xia T. [et al.] Exploring the Effect of Red and Blue on Cognitive Task Performances Frontiers in Psychology 2016.

30. Zhou Q., Flinchbaugh C. Stop Watching Me! Potential Negative Effects of Electronic Monitoring on Employees' Job Performance Academy of Management Proceedings 2023.

31. Browsing Pattern Analysis: What user browsing Patterns Indicate ResearchGate 2024.

32. IEEE Code of Ethics URL: https://www.ieee.org/about/corporate/governance/p7-8.html

33. The Code affirms an obligation of computing professionals to use their skills for the benefit of society URL: https://www.acm.org/code-of-ethics

34. Trouble with big brother: Counterproductive consequences of electronic monitoring through the erosion of leader-member social exchange
URL: https://onlinelibrary.wiley.com/doi/10.1002/job.2748

35. Understanding and supporting personal activity management by IT service workers
URL: https://dl.acm.org/doi/10.1145/1477973.1477976

36. Workplace Surveillance in Canada: A survey on the adoption and use of employee monitoring applications URL: https://onlinelibrary.wiley.com/doi/10.1111/cars.12448

37. Examining workweek variations in computer usage patterns: An application of ergonomic monitoring software
URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0287976

38. A Mousepad Triboelectric-Piezoelectric Hybrid Nanogenerator (TPHNG) for Self-Powered Computer User Behavior Monitoring Sensors and Biomechanical Energy Harvesting
URL: https://www.mdpi.com/2073-4360/15/11/2462

39. Web-browsing patterns reflect and shape mood and mental health
URL: https://www.nature.com/articles/s41562-024-02065-6

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Система моніторингу активності співробітників ІТ-компаній»

на здобуття освітнього ступеня маігстра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Виконав: Дорохін Д.В, ІСДМ-61

Науковий керівник роботи:

Данильченко В.М.

Київ - 2026

Наукова новизна: **Застосування методів машинного навчання для автоматизованого моніторингу активності співробітників ІТ-компаній.**

Об`єкт дослідження: **Активність співробітників ІТ-компаній під час роботи за ПК.**

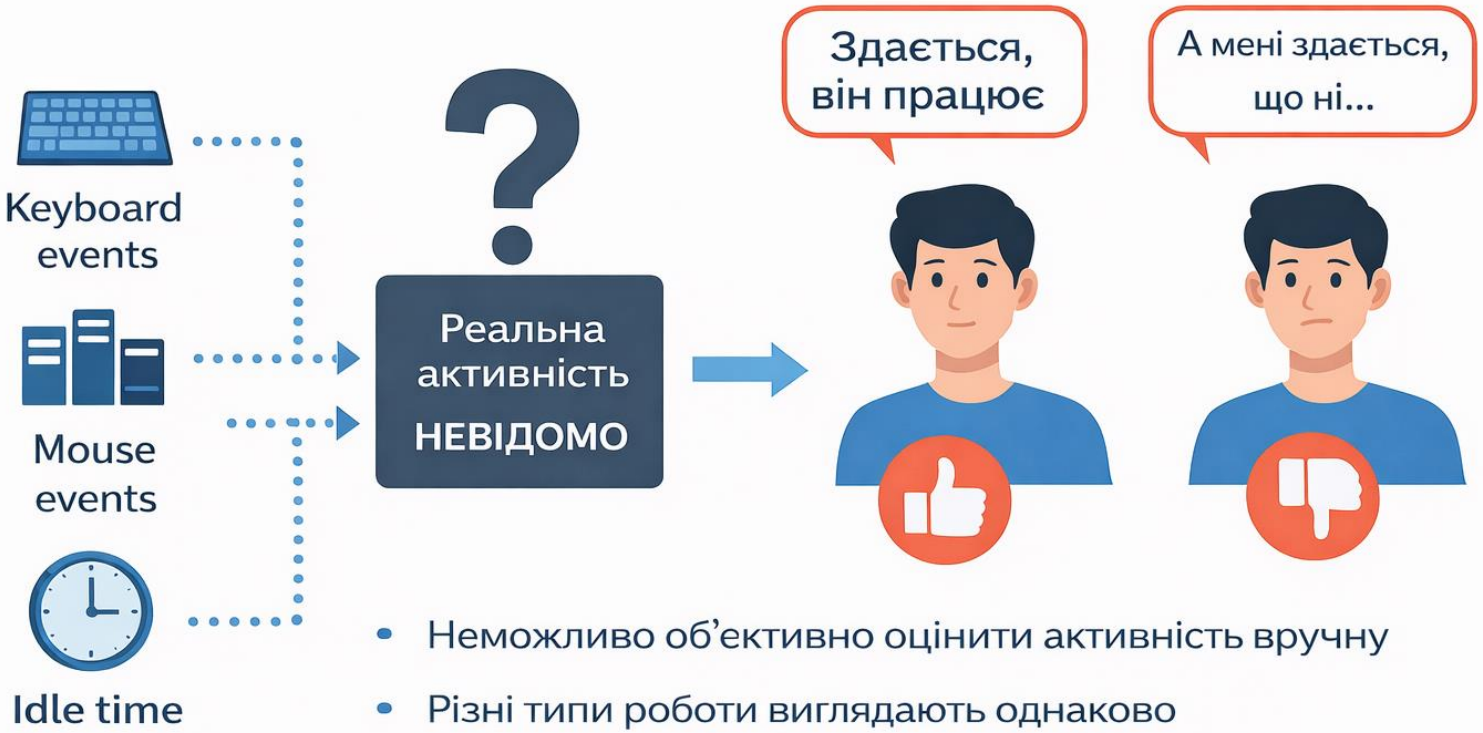
Предмет дослідження: **Автоматизована система моніторингу активності на основі машинного навчання**.

Мета дослідження: **Розробка системи моніторингу активності співробітників ІТ-компаній.**

Завдання дослідження:

1. **Аналіз існуючих систем моніторингу активності.**
2. **Вибір методів машинного навчання для класифікації активності.**
3. **Розробка та тестування автоматизованої системи.**

Проблема оцінки активності співробітників

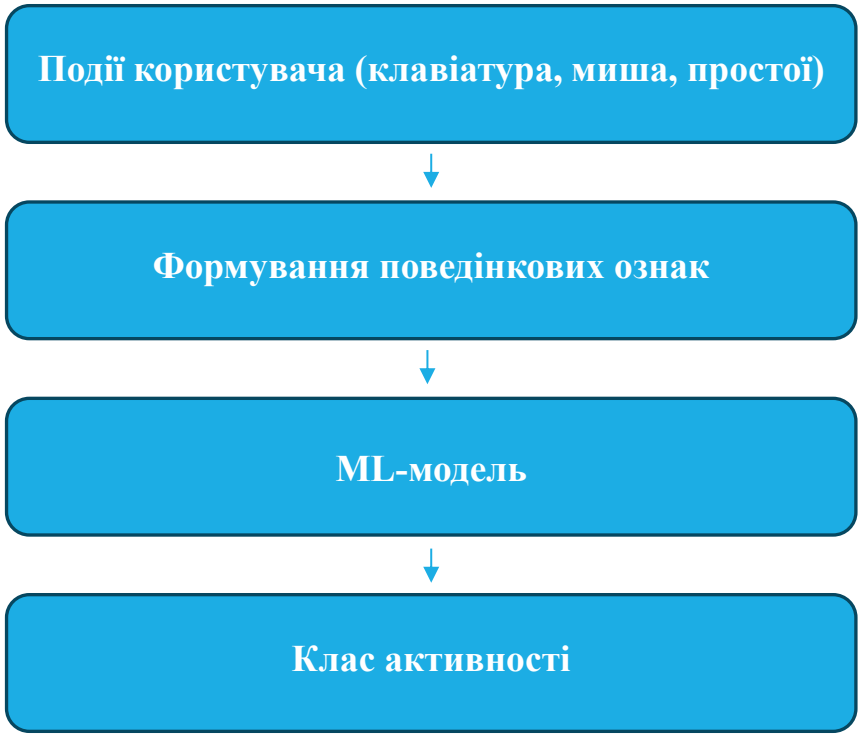


- Неможливо об’єктивно оцінити активність вручну
- Різні типи роботи виглядають однаково
- Суб’єктивні оцінки не відображають реальну діяльність

Аналіз системи

Підхід	Що оцінює	Обмеження
Time-tracking системи	Час роботи	Не показує реальну діяльність
Screen monitoring	Дії на екрані	Інвазивність, відсутність контексту
Правилові системи	Окремі події	Фіксовані пороги, хибні оцінки

Застосування методів машинного навчання для класифікації активності



Запропонований підхід дозволяє перейти від обліку окремих подій до автоматичної класифікації активності користувача.

Збір подій користувача

Події користувача фіксуються на локальному ПК та включають:

натискання клавіш клавіатури

рухи та кліки миші

періоди відсутності активності (простій)

Кожна подія зберігається у вигляді:

мітки часу (timestamp)

коду події (buttonKey)

Події агрегуються у список для подальшого аналізу активності.

Збір подій користувача

Зі списку подій сесії обчислюються такі ознаки:

кількість подій у сесії (count)

середній інтервал між подіями (mean_time_diff)

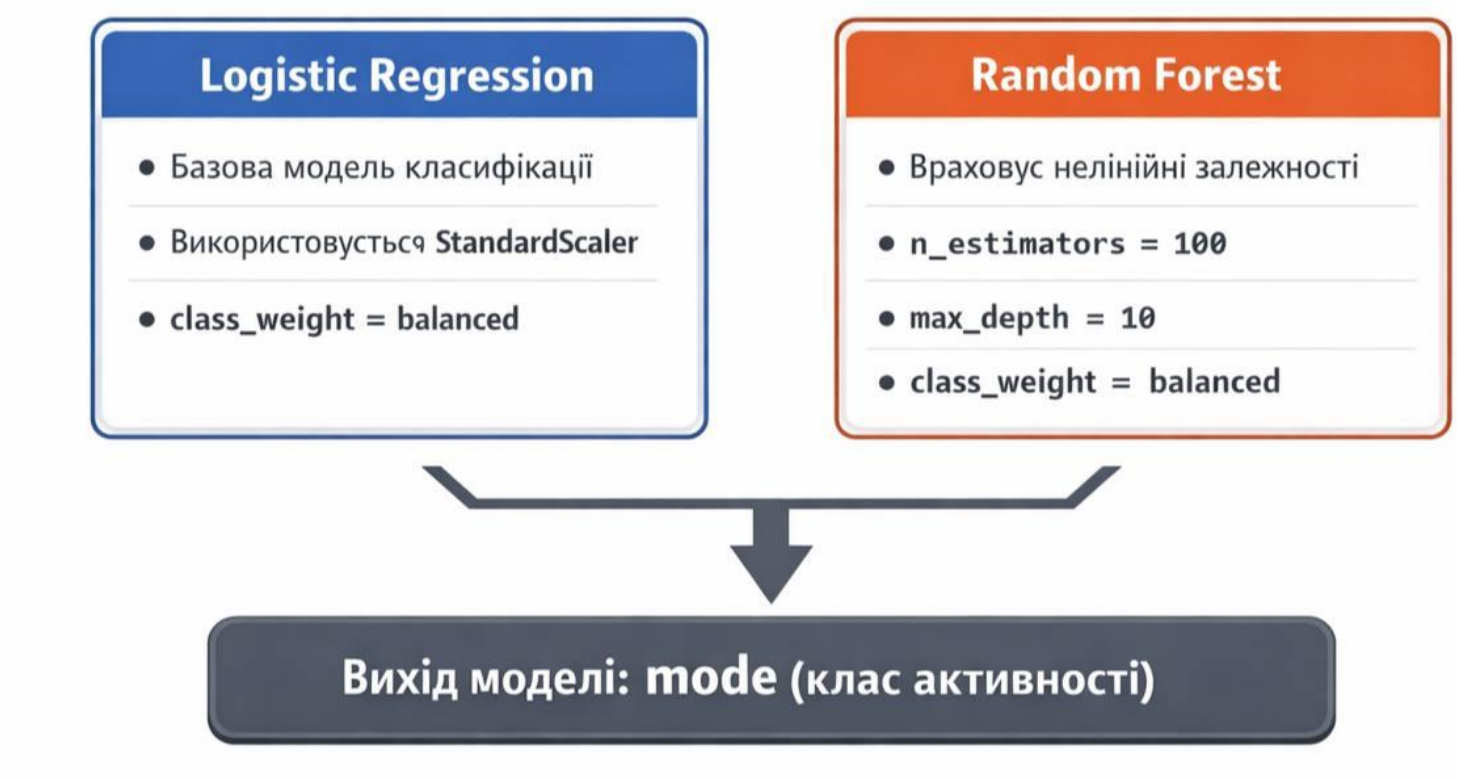
медіанний інтервал між подіями (median_time_diff)

мінімальне та максимальне значення buttonKey

частка подій з малими значеннями buttonKey

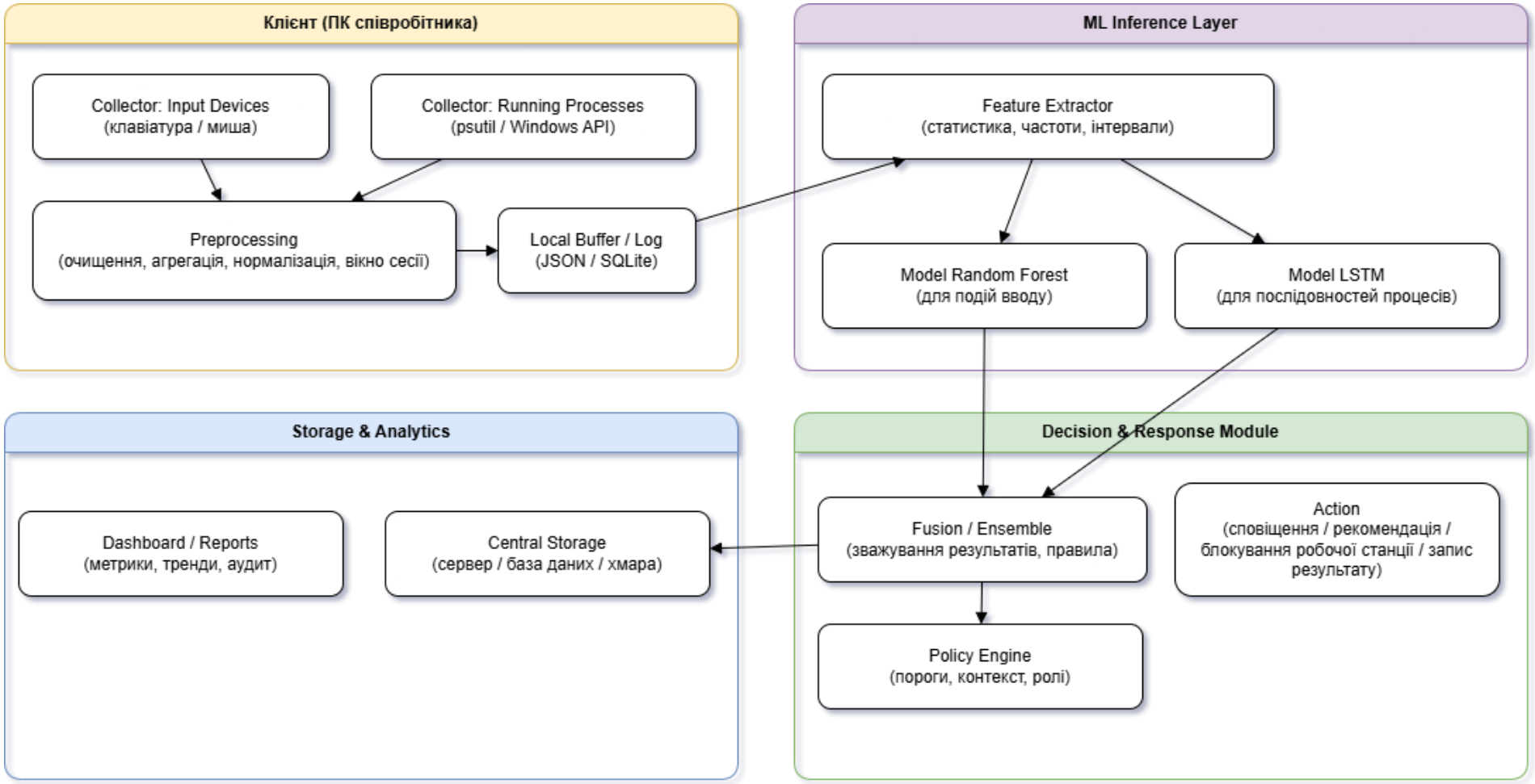
Події попередньо сортуються за timestamp.
У разі відсутності подій обробка не виконується

ML-модель класифікації активності



•У роботі реалізовано два класифікатори: **Logistic Regression** та **Random Forest**
•Для коректної роботи з дисбалансом використано **class_weight = balanced**
•Результат класифікації — значення **mode**, що визначає клас активності

Архітектура системи моніторингу активності



Архітектура системи моніторингу активності

У процесі роботи система автоматично:

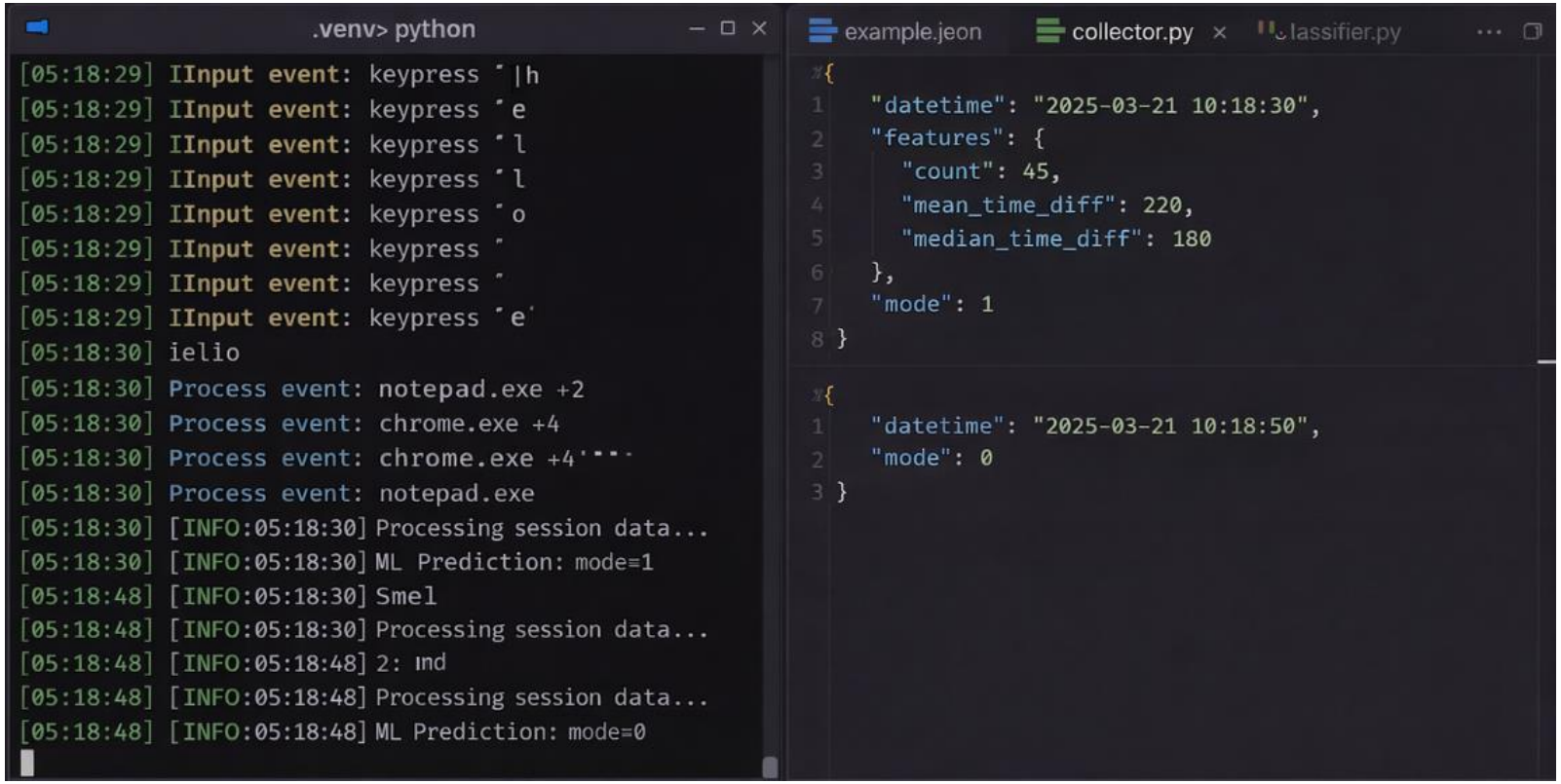
1. формує сесії активності користувача
2. обчислює вектор поведінкових ознак
3. виконує класифікацію за допомогою ML-моделі

Результатом є:

1. визначений клас активності (mode) для кожної сесії
2. збереження результатів для подальшого аналізу

Практичне застосування:

1. об’єктивна оцінка активності
2. аналіз активності у динаміці
3. підтримка управлінських або сервісних рішень



ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено автоматизовану систему моніторингу активності співробітників ІТ-компаній. Запропонована архітектура забезпечує ефективний збір та обробку подій взаємодії користувача з персональним комп’ютером, а також формування поведінкових ознак для подальшої класифікації активності з використанням методів машинного навчання. Реалізований підхід дозволяє автоматично визначати типи активності користувача, здійснювати об’єктивну оцінку робочої діяльності та підтримувати цифровий добробут, що підтверджує практичну цінність і можливість використання розробленої системи для підвищення продуктивності праці.