ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Автоматизація адаптації інтерфейсу під потреби користувача із обмеженими можливостями»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Богдан ЯРОЩУК
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-61
Богдан ЯРОЩУК

Керівник: Тимур ДОВЖЕНКО
канд. техн. наук

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

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ярощуку Богдану Олеговичу

1. Тема кваліфікаційної роботи: «Автоматизація адаптації інтерфейсу під потреби користувача із обмеженими можливостями»

керівник кваліфікаційної роботи Тимур ДОВЖЕНКО, канд. техн. наук,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.

2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, метод кластеризації K-Means, модель прогнозування на основі RNN, вимоги до точності адаптації інтерфейсу.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд предметної області.
2. Аналіз методів кластеризації та моделей штучного інтелекту.
4. Розробка програмного забезпечення моделей.
3. Проведення моделювання та аналіз отриманих результатів.

5. Перелік ілюстративного матеріалу: *презентація*

1. Аналіз існуючих підходів адаптації.
2. Концептуальна модель та алгоритм системи.
3. Збір даних користувача.
4. Вхідна послідовність векторів стану.
5. Механізм оновлення пам’яті в lstm.
6. Оновлення стану та механізми стабілізації.
7. Трансформація ознак та фінальна регресія.
8. Результати навчання моделі LSTM

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.11-06.11.2025	
2	Аналіз стандартів доступності WCAG 2.2 та ISO 9241-171	07.11-09.11.2025	
3	Дослідження технологій штучного інтелекту	10.11-19.11.2025	
4	Розробка блок-схеми роботи додатку	20.11-21.11.2025	
5	Дослідження аріхтектури LTSM	22.11-30.11.2025	
6	Збір та аналіз результатів тестування адаптації інтерфейсу	01.12-14.12.2025	
7	Оформлення роботи: вступ, висновки, реферат	13.12-16.12.2025	
8	Розробка демонстраційних матеріалів	17.12-19.12.2025	

Здобувач вищої освіти _____ Богдан ЯРОЩУК

(підпис)

Керівник
кваліфікаційної роботи _____ Тимур ДОВЖЕНКО

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 92 стор., 1 табл., 29 рис., 34 джерел.

Мета роботи – покращення автоматизованої адаптації інтерфейсів за рахунок прогнозування поведінки користувача за допомогою штучного інтелекту. Це включає створення універсального інструменту, який підвищить доступність цифрових продуктів і технологій, покращуючи взаємодію користувачів із врахуванням їхніх індивідуальних потреб.

Об'єкт дослідження – процес адаптації інтерфейсу цифрових продуктів, які використовуються людьми з обмеженими можливостями.

Предмет дослідження – методи та моделі застосування технологій штучного інтелекту для адаптації інтерфейсів, для людей із порушенням зору

У роботі проведено аналіз сучасних методів адаптації інтерфейсів та вимог міжнародних стандартів доступності. Розроблено та оптимізовано модель автоматизованої адаптації вебінтерфейсу під потреби користувачів із порушенням зору, а також створено прототип браузерного розширення для її практичної реалізації. Проведено експерименти для оцінки точності роботи моделі, ефективності адаптації та відповідності встановленим критеріям доступності.

КЛЮЧОВІ СЛОВА: АДАПТИВНИЙ ІНТЕРФЕЙС, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, РЕКУРЕНТНА НЕЙРОННА МЕРЕЖА, КЛАСТЕРИЗАЦІЯ, ДОСТУПНІСТЬ.

ABSTRACT

Text part of the master's qualification work: 92 pages, 29 pictures, 1 table, 34 sources.

The purpose of the work – is to improve automated interface adaptation by predicting user behavior using artificial intelligence. This includes the development of a universal tool that enhances the accessibility of digital products and technologies, improving user interaction while considering individual needs.

Object of research – the process of adapting the interface of digital products used by people with disabilities.

Subject of research – methods and models of applying artificial intelligence technologies for interface adaptation for people with visual impairments.

Summary of the work: The study includes an analysis of modern methods of interface adaptation and the requirements of international accessibility standards. A model for automated adaptation of web interfaces to the needs of users with visual impairments has been developed and optimized, and a prototype browser extension has been implemented to demonstrate the practical application of the proposed approach. Experiments were conducted to evaluate the accuracy of the model, the effectiveness of adaptation, and compliance with defined accessibility criteria.

KEYWORDS: ADAPTIVE INTERFACE, ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, RECURRENT NEURAL NETWORK, CLUSTERING, ACCESSIBILITY

ЗМІСТ

ВСТУП

Сучасний технологічний прогрес надає можливість людині не лише користуватися численними електронними пристроями та цифровими сервісами, але й інтегрувати їх у повсякденне життя, підвищуючи його якість та зручність. Автоматизовані системи та інформаційні технології значно розширюють межі людських можливостей, зокрема, виконуючи рутинні й ресурсоємні завдання, що потребують точності та швидкості обробки інформації. Водночас, для взаємодії з такими системами розроблено численні інтерфейси, які є важливими засобами комунікації між людиною та технологією.

Згідно зі звітом Global Digital 2023, понад 64% населення світу має доступ до інтернету [1], що підкреслює сьогоднішню потребу у зручних, інтуїтивно зрозумілих інтерфейсах для забезпечення ефективної взаємодії з технологіями. Проте варто наголосити, що значна частина людей з обмеженими можливостями при використанні цифрових продуктів стикаються із бар’єрами, що обмежують їхню участь у суспільному житті та професійній діяльності. За даними Всесвітньої організації охорони здоров'я, близько 15% населення світу мають ті чи інші форми обмежених можливостей, (приблизно 1.3 млрд осіб) [2]. Станом на жовтень 2024 року в Україні офіційно зареєстровано 3 мільйони людей з інвалідністю, з яких приблизно 300 тисяч набули цього статусу внаслідок повномасштабного вторгнення росії [3]. Тому в Україні, потреба в технологіях для підтримки інтеграції таких осіб в суспільство та професійне середовище є особливо актуальною.

Інтерфейси, що адаптуються під індивідуальні потреби користувачів, мають на меті зменшення цифрової нерівності та забезпечення доступності цифрових продуктів для кожного [4].

Персоналізовані адаптивні інтерфейси на базі штучного інтелекту (ШІ) забезпечать доступ до цифрових сервісів для цих людей [2], полегшать їхню інтеграцію в суспільне життя та професійну діяльність, тим самим допомагаючи державі у післявоєнному відновленні.

Використання ШІ у створенні таких інтерфейсів відкриває нові можливості для персоналізації взаємодії користувачів з цифровими продуктами, враховуючи їхні індивідуальні можливості та потреби [5]. Це включає в себе розробку графічних, голосових, тактильних інтерфейсів де машинне навчання дозволяє системам аналізувати поведінку користувача та персоналізовувати інтерфейс відповідно до його потреб. Такий підхід забезпечить рівний доступ до технологій для всіх користувачів, незалежно від індивідуальних особливостей.

Для українських ветеранів та осіб які набули фізичні обмеження в наслідок бойових дій, адаптивні інтерфейси надають можливість оволодіти новими професіями чи використовувати цифрові технології для створення власного джерела доходу, що також матиме позитивний психологічний вплив, оскільки сприятиме відчуттю соціальної значущості та необхідності. Це дозволить не лише реалізувати їхній професійний потенціал, але й інтегруватися в суспільство як активні його учасники, що підвищить рівень їхньої самооцінки та емоційного благополуччя.

Впровадження таких технологій не тільки в Україні а й у світі допоможе створити цифрове середовище, яке відповідає потребам постраждалих від війни та сприяє інтеграції людей з інвалідністю в цифровий простір роблячи його більш доступним та відкритим, тим самим наблизитись до глобальної мети зменшення цифрової нерівності та забезпечення доступності цифрових продуктів для кожного [4].

Мета дослідження - покращення автоматизованої адаптації інтерфейсів за рахунок прогнозування поведінки користувача за допомогою штучного інтелекту. Це включає створення універсального інструменту, який підвищить доступність цифрових продуктів і технологій, покращуючи взаємодію користувачів із врахуванням їхніх індивідуальних потреб.

Гіпотеза дослідження - використання адаптивних інтерфейсів, створених на основі штучного інтелекту, здатне суттєво покращити взаємодію людей з обмеженими можливостями з цифровими продуктами, підвищуючи їхню доступність та ефективність інтеграції у соціальне і професійне життя.

Об'єкт дослідження – процес адаптації інтерфейсу цифрових продуктів, які використовуються людьми з обмеженими можливостями.

Предмет дослідження – методи та моделі застосування технологій штучного інтелекту для адаптації інтерфейсів, для людей із порушенням зору.

Завдання дослідження:

1. Провести аналіз існуючих рішень у сфері адаптивних інтерфейсів для людей з обмеженими можливостями, визначити їх переваги, недоліки та рівень доступності.
2. Розробити концепцію адаптивного інтерфейсу, що враховує індивідуальні потреби користувачів з обмеженими можливостями.
3. Використати алгоритми штучного інтелекту (нейронні мережі, машинне навчання) для персоналізації взаємодії користувача з цифровим продуктом.
4. Запропонувати прототип браузерного розширення, який автоматично адаптує інтерфейс веб-сайтів для користувачів із порушенням зору.
5. Провести тестування запропонованого рішення з метою визначення його ефективності та зручності використання.

Методи дослідження - застосування рекурентної нейронної мережі для прогнозування і моделювання поведінки користувачів, шляхом збору та класифікації даних про взаємодію користувачів із веб-інтерфейсами із використанням алгоритмів машинного навчання зокрема кластеризації на основі K-Means. Зібрані та класифіковані дані будуть використані рекурентною нейронною мережею (RNN) для аналізу часових послідовностей змін поведінки користувачів і динамічної адаптації інтерфейсу у реальному часі. Навчена нейронна мережа, здатна в реальному часі аналізувати поведінку користувача та прогнозувати необхідні зміни в інтерфейсі для підвищення зручності взаємодії.

Автоматизація використаних нейронних мереж шляхом розробки прототипу браузерного розширення яке на основі зібраних даних про забезпечує адаптований веб-інтерфейс під потреби користувача із проблемами зору.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ І СУЧАСНИЙ СТАН ТЕХНОЛОГІЙ ДЛЯ АДАПТИВНИХ ІНТЕРФЕЙСІВ

1.1 Визначення поняття адаптивних інтерфейсів та їх значення для людей з обмеженими можливостями

Термін "інтерфейс", який походить від англійського слова "interface" — "поверхня розділу" або "перегородка", спочатку використовувався в хімії та фізиці для опису межі між різними середовищами або фазами [6]. Однак з розвитком комп'ютерних технологій значення цього терміну поширилось і на галузь інформаційних технологій, де "інтерфейс" став означати засоби та принципи взаємодії між користувачем і пристроєм [5]. На ранніх етапах комп'ютеризації взаємодія з комп’ютером вимагала наявність кваліфікованих фахівців для програмування машини на виконання певного завдання. Наприклад, перший програмований комп’ютер ENIAC потребував цілої команди інженерів для обслуговування та налаштування, оскільки процес перепрограмування передбачав фізичну перебудову панелей та переміщення кабелів.

Все змінилося 9 грудня 1968 року, коли науковець Дуглас Енгельбарт зі Стенфорду продемонстрував перші ідеї графічного інтерфейсу користувача. Ця ідея суттєво вплинула на розвиток сучасних операційних систем, таких як Windows чи macOS, і дала поштовх еволюції інтерфейсів — від командного рядка до зрозумілих, візуальних та інтерактивних елементів.

Інтерфейси еволюціонували разом із розвитком технологій, перетворившись із простих текстових командних систем на інтуїтивно зрозумілі, інтерактивні графічні елементи. Цей розвиток став відповіддю на потребу користувачів у зручності та ефективності взаємодії з пристроями.

Сьогодні розрізняють два основні типи інтерфейсів: адаптивні та адаптовані. Зовні вони схожі, але їхні функціональні принципи суттєво відрізняються:

Адаптивний інтерфейс — це система, яка самостійно підлаштовується під потреби користувача. Вона змінює вигляд і поведінку залежно від того, як, коли і де користувач взаємодіє з продуктом. Наприклад, інтерфейс може автоматично перемикатися в нічний режим, змінювати масштаб елементів чи пропонувати голосовий супровід.

Адаптований інтерфейс — це той, що потребує ручного налаштування. Користувач сам змінює параметри відповідно до своїх потреб — змінює шрифт, кольори, контрастність тощо. Система в цьому разі лише надає інструменти, але не виконує змін автоматично. Це дозволяє створити індивідуалізований досвід, проте вимагає більше зусиль з боку користувача [9].

Обидва типи інтерфейсів спрямовані на підвищення доступності, функціональності та комфортності у взаємодії з цифровими продуктами, однак їхні підходи до досягнення цієї мети суттєво відрізняються, що особливо важливо для людей з обмеженими можливостями.

Щоб зрозуміти це, уявіть просту ситуацію: кожного ранку ви налаштовуєте розмір тексту, прибираєте зайві зображення, змінюєте кольори — тільки для того, щоб прочитати новини. І це повторюється знову і знову на кожному новому сайті або додатку. Скільки часу це займатиме? Як це вплине на ваш користувацький досвід, якщо такі налаштування доведеться виконувати кожного разу при використанні нового цифрового продукту?

А тепер уявіть іншу ситуацію: ви відкриваєте смартфон, і все вже зручно підібрано під вас — великий шрифт, потрібна кольорова гама, зайве приховано, а голосовий помічник пропонує озвучити текст, поки ви займаєтесь іншими справами. Все відбулося автоматично, без жодного зусилля з вашого боку. Це і є приклад справжнього адаптивного інтерфейсу, який забезпечує комфортну взаємодію.

Люди з обмеженими можливостями щодня стикаються з подібними викликами, які часто залишаються непоміченими для більшості користувачів. Для них навіть найпростіша взаємодія з цифровим середовищем може стати бар’єром, якщо інтерфейс не враховує їхніх потреб. Що вона відчуває коли вона

самостійно не може отримати послугу чи скористатись цифровим інструментом через банальну незручність чи не адаптований інтерфейс?

Тому значення адаптивних інтерфейсів є без перебільшення величезним. Такі технології є важливим кроком у забезпеченні створення інклюзивного цифрового простору, де кожен користувач зможе відчувати комфорт і автономність, незалежно від фізичних або когнітивних обмежень. Вони не лише підвищують зручність у користуванні, а й сприяють соціальній інтеграції, даючи доступ людям із різними потребами до спілкування, освіти, праці та розваг — і тим самим наближають нас до справжньої цифрової рівності.

1.2 Сучасні технології та підходи до адаптивних інтерфейсів із використанням ШІ

З кожним роком цифрові продукти стають дедалі складнішими й багатофункціональнішими, вони працюють із великими обсягами даних і обслуговують мільйони користувачів одночасно. Але зі зростанням користувачів і зростає різноманіття потреб та сценаріїв використання цифрових продуктів. Тому не менш важливо не лише створювати гарний дизайн, а й зробити так, аби ці інтерфейси могли пристосовуватись до різних потреб користувача. Саме на цьому етапі технології штучного інтелекту починають відігравати головну роль. Вони здатні дати системам можливості не бути «статичними», а навпаки — гнучко реагувати на ситуацію, врахувати навколишнє середовище, поведінку та особливості людини, з якою взаємодіють. Вони можуть забезпечити самонавчання для персоналізації для кожного конкретного випадку.

Однією з ключових технологій, яка активно використовується в адаптивних системах, є машинне навчання. Це підхід, за якого програма аналізує велику кількість даних та поведінку користувача, які кнопки натискає, як часто звертається до тієї чи іншої функції, наскільки довго затримується на одному чи іншому блоці, які елементи прибирає тощо. На основі цих даних, алгоритм робить висновки — складає, так би мовити, паттерн поведінки користувача. На

базі цього паттерну алгоритм поступово «вчиться» підлаштовуватись — приховувати непотрібне, підкреслювати важливе, змінювати розташування елементів, змінювати розмір кнопок тощо. Наприклад, Netflix використовує машинне навчання не тільки для рекомендацій фільмів. Якщо в користувача виявляються труднощі зі сприйняттям складних інтерфейсів (наприклад, у людей з когнітивними порушеннями або аутизмом), система може автоматично спростити вигляд головної сторінки — зменшити кількість обкладинок, прибрати анімацію, зробити крупніші заголовки. Це допомагає не відволікати увагу й зменшує когнітивне навантаження. Аналогічно працює і Google Maps — платформа не лише пропонує маршрути відповідно до вподобань користувача, але й адаптує свій вигляд для людей із порушенням зору, додаючи докладні голосові підказки. Окрім цього підбирає маршрути на основі попередніх поїздок чи прогулянок враховуючи складні перехрестя чи активний трафік.

Не менш впливовим напрямом є технології обробки природної мови (Natural Language Processing, NLP). Вона дає змогу інтерфейсам «розуміти» людську мову, не просто реагувати на стандартні голосові команди, а й розуміти контекст розмови, розуміти “живу” мову та навіть адаптувати свою манеру спілкування до манери користувача. У цьому контексті хорошим прикладом є ChatGPT: він здатен вести діалог навіть із тими, хто має труднощі з письмовою комунікацією. Система автоматично налаштовує стиль відповіді — спрощує її або навпаки, деталізує — в залежності від того, як сформульоване запитання. Система дає універсальність, що дає змогу використовувати її і школярам, і вчителям, і людям із різними мовленневими розладами. Інши приклад — Google Assistant не просто виконує команду «увімкни світло», а може зрозуміти синонімічні фрази на кшталт «мені тут темно» або «зроби яскравіше». Більше того, він здатен підлаштовувати темп, інтонацію та навіть словниковий запас до стилю мовлення користувача. Якщо людина говорить повільно або з акцентом — система знижує швидкість відповіді або додає паузи.

Окремої уваги заслуговує технологія комп’ютерного зору. Вона дозволяє інтерфейсу буквально «бачити» — тобто аналізувати візуальні дані такі як

зображення, відео або навіть просторові об’єкти в реальному часі і реагувати на них. Наприклад, Face ID від Apple значно полегшує доступ до пристроїв для людей, які не можуть користуватись руками. Система розпізнає обличчя та автоматично відкриває доступ до пристрою. А Google Lens допомагає тим, хто має порушення зору, розпізнаючи об’єкти чи текст на зображеннях і озвучуючи їх. Система здатна в реальному часі розпізнавати, перекладати, озвучувати і зберігати текст. Все що вам для цього потрібно камера смартфону. Технологія не просто виконує функцію, а розширює можливості людини.

Дуже перспективна але і дуже чутлива технологія — розпізнавання емоцій. Завдяки аналізу міміки, тону голосу, ритму мовлення або навіть стилю листування, система може зрозуміти, в якому емоційному стані перебуває користувач, і підлаштувати інтерфейс під нього. Ідея полягає у тому щоб система могла виявляти, коли користувач стомлений, а коли зацікавлений. Microsoft Azure Emotion API, наприклад, змінює поведінку програми, якщо виявляє тривожний або пригнічений стан. А додаток Replika йде ще далі — він не просто веде діалог, а намагається бути «чутливим» до стану користувача, особливо якщо той переживає психологічні труднощі. Це відкриває нові горизонти у створенні емпатичних інтерфейсів, здатних підтримати людину не лише функціонально, а й морально. Чутливість та етична сторона даної технології полягає у питанні чи не використають компанії такий інструмент щоб продавати свої товари чи послуги? Наприклад реклама відповідного товару буде показуватись тільки тоді коли людина буде максимально схильна купити, чи послуги будуть пропонуватись у моменти емоційного збудження чи навпаки. Іншими словами максимально персоналізована реклама яка буде змушувати вас купувати той товар чи послугу відповідно до емоційного стану. Такий інструмент буде доступний лише великим корпораціям, і як наслідок — менші компанії, позбавлені подібних технологічних можливостей, опиняться у нерівних умовах конкуренції, що може призвести до їх поступового витіснення з ринку.

Етична сторона полягає у необхідності встановлення чітких меж між корисною персоналізацією та маніпуляцією поведінкою користувача. Важливо,

щоб алгоритми не перетворювалися на інструменти впливу на підсвідомість чи емоційний стан людини. Для цього необхідні прозорі механізми контролю, етичні стандарти розробки та законодавче регулювання. Лише за таких умов адаптивні системи можуть залишатися засобом покращення якості життя, а не інструментом комерційного тиску.

Поза конкретними технологіями, важливим є й те, як саме система вирішує адаптуватися. Тут варто розглянути підходи, до самого процесу адаптації, тобто як система вирішує змінювати свій вигляд або поведінку.

Почнемо із адаптації яка орієнтована на користувача (User-Centered Adaptation). Цей підхід ставить у центр уваги саму людину — її вік, рівень знань, фізичні особливості, стиль взаємодії або навіть стан здоров'я. Система збирає ці дані, "вивчає" їх та підлаштовує інтерфейс чи подачу контенту відповідно до індивідуальних характеристик.

До цього ж напрямку належить і адаптації на основі поведіки (Behavior-Based Adaptation). адже обидва підходи тісно пов'язані між собою: у першому враховуються особистісні параметри користувача, а в другому — його дії, реакції та шаблони використання системи.

Яскравим прикладом є сервіс Netflix який формує індивідуальні рекомендації, враховуючи, які фільми ви переглядаєте, як довго тримаєте увагу, які жанри оцінюєте, як реагуєте. Якщо користувач часто перериває перегляд драм, але залишається до кінця на комедіях — платформа пропонує більше легкого контенту. Amazon діє подібно — вивчає, що ви шукаєте, на що клікаєте, що додаєте у вішлисти, що купуєте. На основі цієї поведінки система не просто показує товари, а «вгадує» ваші інтереси, наприклад: якщо ви купили телефон, алгоритм автоматично пропонує чохли, захисне скло чи зарядний пристрій. Duolingo іде ще далі — рівень складності вправ підлаштовується під користувача. Якщо завдання даються легко то з'являються складніші. Якщо помиляєтесь — система повертається до простішого матеріалу. Facebook, адаптує вашу стрічку новин на основі того як ви з яким раніше взаємодіяли, також береться до уваги взаємодія із контентом ваших друзів.

Якщо фокус зміщується з особистості користувача на тип завдання, яке він виконує, така адаптація орієнтована на завдання (Task-Based Adaptation). Інтерфейс змінюється залежно від контексту дій, етапу виконання завдання, складності чи необхідних інструментів. Мета такого підходу — спростити взаємодію користувача із системою, зменшити кількість непотрібних елементів на екрані та забезпечити швидкий доступ лише до тих функцій, які є актуальними в даний момент. У системах проєктування (CAD) інтерфейс може автоматично змінювати набір інструментів залежно від того, чи користувач працює над кресленням, 3D-моделюванням або візуалізацією. Microsoft Word це класичний приклад, коли ви редагуєте таблицю, активуються тільки ті вкладки й функції, які стосуються роботи з таблицями. Adobe Photoshop та Visual Studio Code як приклади професійних програм у яких адаптація працює залежно від типу проєкту. Якщо це монтаж, анімація чи кодинг, відповідний набір інструментів буде у швидкому доступі та автоматично налаштовані під задачу. З таким підходом отримуємо пришвидшення роботи та зменшення відволікання та часу на знаходження потрібного інструменту.

Адаптація залежно від пристрою (Device-Aware Adaptation) варта уваги так як користувач обирає для користування доступний та найбільш комфортний для себе пристрій. У такому випадку система має враховувавати технічні характеристики пристрою: розмір екрана, продуктивність, тип операційної системи, тип введення даних тощо, і відповідно змінювати свій інтерфейс і функціональність. На даний момент такий підхід до адаптації є одним із ключових напрямів розвитку інклюзивних технологій. Наприклад, вебзастосунки автоматично змінюють розташування елементів для зручності користування на смартфонах або планшетах, а мобільні операційні системи можуть спрощувати візуальні ефекти для зниження навантаження на процесор. В якості прикладів використання такого підходу можна навести YouTube який автоматично змінює якість відео залежно від швидкості інтернету та продуктивності пристрою. Сервіс змінює інтерфейс відповідно до типу пристрою — на ПК показує повну версію інтерфейсу з коментарями, плейлистами, статистикою. На смартфоні

інтерфейс максимально спрощено — крупні кнопки, швидкий доступ до відео, без зайвих елементів. Ще декілька прикладів: Microsoft Teams — знижує роздільну здатність відео або вимикає ефекти фону на малопотужних пристроях. У грі Fortnite графіка й управління змінюються залежно від платформи: на мобільному — простіша візуалізація й сенсорне керування, на ПК — розширені опції та деталізована графіка. Google Maps на телефоні й в авто поводиться по-різному: у смартфоні — докладна карта, панелі інструментів, відгуки; в Android Auto — лише голосові інструкції й великі кнопки для безпечної навігації.

Наступний напрям є адаптація на основі часу (Time-Based Adaptation). Система реагує на час доби, день тижня, тривалість активності користувача чи частоту його дій, щоб змінювати інтерфейс, функціональність або рівень навантаження. Наприклад операційні системи iOS і Android автоматично перемикають інтерфейс на темний режим у вечірній час або при слабкому освітленні. Новинна платформа Flipboard, змінює пріоритетність новин: зранку — головні події, вдень — бізнес, увечері — легкий контент.

Наступний підхід який розглянуто адаптація орієнтована на соціальний контекст (Social-Aware Adaptation). Цей тип адаптації враховує соціальне оточення користувача, його взаємодію з іншими людьми, належність до певних спільнот або груп, а також контекст спільної діяльності. Система аналізує соціальні сигнали — спільне використання документів, участь у відеоконференціях, комунікацію в месенджерах чи соціальних мережах — і на основі цього підлаштовує інтерфейс або пропонований контент. Наприклад, у Microsoft Teams інтерфейс може змінюватися залежно від того, чи бере користувач участь у груповому проєкті або проводить індивідуальну зустріч. У спільному режимі з’являються інструменти для колективного редагування файлів, обміну коментарями чи спільного планування. У соціальних мережах, як-от Facebook або LinkedIn, алгоритми формують стрічку новин з урахуванням соціальних зв’язків — частіше показують пости тих, з ким користувач активно взаємодіє. Такий підхід сприяє покращенню командної роботи, ефективнішій комунікації та підвищенню залученості користувачів, однак водночас викликає

питання щодо конфіденційності та етичності збору даних, адже аналіз соціальних зв’язків потребує обробки великої кількості персональних даних.

Адаптація, заснована на часі (Time-Based Adaptation). У цьому підході система враховує часові фактори — добу, частоту використання, тривалість активності користувача чи динаміку його дій. Відповідно до цих параметрів інтерфейс або функціональність можуть змінюватися, щоб підтримувати комфортну взаємодію. Наприклад, мобільні операційні системи автоматично вмикають нічний режим після заходу сонця, знижуючи яскравість і змінюючи кольорову гаму для зменшення втоми очей. У фітнес-додатках інтерфейс змінюється залежно від часу тренування — вранці можуть з’являтися нагадування про розминку, а ввечері — рекомендації для відновлення. Сервіси на кшталт Google Calendar або Todoist також адаптують повідомлення залежно від завантаженості дня, пропонуючи оптимальний час для відпочинку або виконання завдань.

Останній розглянутий підхід це адаптація за правилами (Rule-Based Adaptation). Це класичний підхід, коли адаптація не відбувається автоматично, заданих правил, визначених розробником, або адміністратором, або налаштовано користувачем. У цьому випадку система не “вчиться” самостійно, а реагує на заздалегідь передбачені сценарії. Наприклад, Nest Thermostat автоматично знижує температуру, якщо нікого немає вдома, або вмикає обігрів, коли показники падають нижче встановленого рівня. На платформі Khan Academy, якщо учень кілька разів поспіль відповідає правильно, завдання поступово ускладнюються, а у разі помилок — система повертає до базового рівня. У системах контролю доступу подібна логіка теж застосовується: співробітники можуть увійти до будівлі лише у визначені години. А додаток Kidslox блокує контент, що не відповідає віковим обмеженням.

Як узагальнення для вище сказаного можемо сказти, що поєднання технологій штучного інтелекту з глибоким розумінням людських потреб дозволяє створювати адаптивні інтерфейси, які не просто «працюють», а працюють для конкретної людини — гнучко, чутливо, персоналізовано. Сучасні

тенденції свідчать, що найкращих результатів можна досягти не завдяки окремому методу, а поєднанню кількох підходів одночасно. Використання технологій штучного інтелекту відіграє ключову роль у цьому процесі, оскільки дає змогу системам навчатися, прогнозувати поведінку користувача, адаптувати інтерфейс у реальному часі та враховувати індивідуальні особливості кожної людини.

Поєднання різних типів адаптації з алгоритмами відкриває нові можливості для створення інклюзивних рішень, які допоможуть людям з обмеженими можливостями повноцінно взаємодіяти з цифровими продуктами, навчатися, працювати та реалізовувати свій потенціал у суспільстві. Такий напрям розвитку сприятиме формуванню доступного, гнучкого й гуманного цифрового середовища, у якому технології служать людині незалежно від її фізичних чи когнітивних особливостей.

1.3 Переваги та недоліки існуючих рішень

Розвиток адаптивних інтерфейсів є важливим кроком у напрямі побудови інклюзивного цифрового середовища. Проте, як і кожна технологічна інновація, ці рішення мають не лише сильні сторони, а й певні недоліки, які обмежують їх ефективність і універсальність. У цьому розділі детально розглянемо обидві сторони цього явища — з практичними прикладами, які допоможуть краще зрозуміти реальні можливості та виклики адаптивних систем.

Переваги адаптивних інтерфейсів:

Однією з ключових переваг таких рішень є підвищення доступності цифрових продуктів для користувачів з інвалідністю. Завдяки вбудованим функціям адаптації, люди з порушенням зору, слуху, опорно-рухового апарату чи когнітивними особливостями можуть ефективніше користуватися цифровими сервісами. Наприклад, YouTube надає функцію автоматичної генерації субтитрів, за 2023 рік ця функція покрила понад 1 мільярд годин відео, що зробило контент доступним мільйонам глядачів з порушеннями слуху не зважаючи на те що

генерація не завжди досконала. Інший приклад — Microsoft Immersive Reader, який інтегрований у Word, Edge та OneNote. Інструмент який дозволяє змінювати структуру тексту, колірну палітру, розмір шрифтів, а також озвучувати текст, що надзвичайно корисно для людей з дислексією або слабким зором регулярно використовують 35 мільйонів користувачів щомісяця, переважно — школярі та студенти з дислексією.

Ще однією суттєвою перевагою є високий рівень персоналізації, який забезпечується завдяки використанню алгоритмів штучного інтелекту. Вони дозволяють інтерфейсам підлаштовуватися до вподобань, звичок і навіть настрою користувача. Наприклад, Netflix аналізує, які жанри фільмів ви переглядаєте найчастіше, скільки часу витрачаєте на пошук, і пропонує добірки, які найбільше відповідають вашим смакам. У 2022 році Netflix повідомив, що 80% переглядів на платформі припадають на рекомендований контент, сформований адаптивними алгоритмами. Інтерфейс Google Chrome, у свою чергу, дозволяє змінювати кольорову гаму, масштабування, ввімкнути режим підвищеної контрастності, що зручно не лише для користувачів з особливими потребами, а й для будь-кого, хто працює в складних умовах освітлення. Схожі підходи застосовують Spotify (адаптивні плейлисти), Amazon (персоналізовані добірки товарів), середня конверсія в інтернет-магазинах із персоналізованим інтерфейсом може бути до 30% вищою, ніж у традиційних.

Адаптивні інтерфейси також суттєво покращують загальний користувацький досвід (UX). Наприклад, Apple впровадила функцію True Tone — адаптацію температури кольору екрана під освітлення навколо. Це не лише комфортно, але й зменшує втомлюваність очей. За результатами внутрішніх опитувань компанії, понад 85% користувачів позитивно оцінили цю функцію. Android має подібну опцію — адаптивну яскравість і нічний режим. А в автомобільних системах — як-от Tesla UI — елементи автоматично масштабуються під час руху, враховуючи безпечне сприйняття на ходу. У деяких смартфонах адаптація здійснюється також на основі місця перебування чи навіть

рівня шуму — наприклад, пристрій автоматично вмикає режим «Не турбувати», коли користувач перебуває у кінотеатрі.

Інклюзивність, як цінність, закладена в основі адаптивного дизайну, дозволяє створювати такі продукти, які не виключають жодної категорії користувачів. Google Maps, наприклад, має функції голосової навігації з детальними інструкціями щодо поворотів, зупинок, зміни рівня дороги — усе це допомагає людям із порушенням зору безпечно переміщатися. За рік понад 2 мільйони користувачів активували цю функцію. А Microsoft Teams запровадив підтримку жестової мови під час відеодзвінків, що дало змогу людям з порушенням слуху брати активну участь у роботі та навчанні. Обидва приклади інтерфейсів не просто допомагають — вони створюють відчуття рівноцінної участі незважаючи на індивідуальні особливості чи бар'єри, які раніше могли заважати повноцінній взаємодії.

Крім цього, адаптивні інтерфейси добре інтегруються з допоміжними технологіями. Наприклад, Apple VoiceOver озвучує всі дії на екрані, дозволяючи людям із повною втратою зору керувати смартфоном. VoiceOver безкоштовне, вбудоване рішення, яке у 2023 році використовували понад 250 мільйонів людей у світі. Аналогічно, програмне забезпечення JAWS (Job Access With Speech) дозволяє користувачам із вадами зору працювати з ПК, прослуховуючи вміст екрана й керуючи інтерфейсом за допомогою клавіатури. JAWS популярний екранний зчитувач, незважаючи на ціну ліцензіїї (для домашнього використання ціна місячної підписки – 98$, вартість пожиттєвої – 1548$) використовується у понад 60 країнах, що може свідчити про професійний рівень інструменту і його затребуваність.

Недоліки та виклики:

Водночас існує ряд проблем, які не дозволяють адаптивним інтерфейсам повністю реалізувати свій потенціал. Насамперед, це обмежена універсальність. Багато систем добре працюють для однієї категорії користувачів, але виявляються погано адаптованими або навіть незручними для інших. Наприклад, у 2023 році лише 15% найпопулярніших мобільних ігор у App Store мали базові

опції доступності. Instagram підтримує автоматичні субтитри, але їх точність, особливо в мовах, де розпізнавання не оптимізоване для неангломовних користувачів досі залишається низькою. Подібна ситуація і з відеоіграми — більшість популярних проєктів, як-от Fortnite або Call of Duty, не передбачають адаптивних налаштувань для гравців із порушенням моторики чи зору, хоча потенційна аудиторія таких користувачів — мільйони.

Інша серйозна проблема — висока вартість впровадження. Створення однієї повноцінної адаптивної системи може коштувати від $50,000 до $250,000 — залежно від складності, кількості мов, рівня персоналізації й технічного стека. Адаптивні рішення не є «дешевими в обслуговуванні»: вони вимагають постійного аналізу даних, оновлення моделей ШІ, тестування на реальних користувачах. Також це все вимагає створення та підтримки складної інфраструктури такої як сервери для обробки даних, навчання моделей тощо. Наприклад, підтримка Live Captions у Google Meet вимагає постійної обробки мови в реальному часі, що реалізується на потужних серверах, витрачаючи ресурси навіть на короткі розмови. Платформи малого та середнього бізнесу часто просто не мають ресурсів на впровадження таких рішень.

Також спостерігаються проблеми сумісності, особливо на старих або малопотужних пристроях. У світі понад 40% людей користуються пристроями, випущеними понад 5 років тому. На таких пристроях навіть базові адаптивні функції працюють з перебоями. Наприклад, функція голосового введення в Google Docs може працювати некоректно, якщо комп’ютер не підтримує новітні аудіодрайвери або має слабкий процесор, що може викликати втрату точності. У Microsoft Teams іноді спостерігається затримка в обробці відео або голосу, навіть на більш сучасних пристроях, що ускладнює спілкування в реальному часі та може бути особливо критичним для навчального процесу чи онлайн-роботи. І виникає така ситуація, що навіть високоякісні рішення втрачають свою ефективність, коли користувач змушений працювати на застарілому обладнанні. Це створює приховану нерівність між тими, хто має доступ до сучасних технологій, і тими, хто користується старими пристроями. У результаті

отримуємо замкнене коло, що доступність до цифрових сервісів стає не лише питанням програмного забезпечення, але й технічних можливостей пристрою конкретного користувача.

Наступна проблема яка варта уваги — недостатня локалізація. Україна — одна з країн, які страждають від цифрової нерівності в плані мовної підтримки. Так, у 2023 році YouTube автоматично генерував субтитри українською лише до 12% відео, навіть якщо вони були записані українською мовою. Багато навчальних сайтів — як-от Khan Academy — підтримує менш ніж 10 мов, серед яких українська відсутня, попри активну аудиторію з України, через що їхні адаптивні функції стають недоступними для більшості населення світу.

Не менш критичним є і питання конфіденційності. Збір та захист даних — одна з чутливих і найсуперечливіших тем. Адаптивні системи майже завжди потребують великих обсягів персональних даних: геолокація, історія переглядів, активність у соціальних мережах, емоційні реакції тощо. Алгоритми Google Assistant чи Meta часто обробляють ці дані для підвищення персоналізації, але користувачі рідко розуміють, як саме використовується ця інформація, хто до неї має доступ і які ризики вона несе. Часто користувачі навіть не знають яку інформацію додаток збирає і чи збирає взагалі, так наприклад у 2022 році Google Assistant отримав скарги від понад 100 тисяч користувачів через те, що активувався без запиту. Алгоритми Meta (Facebook) часто адаптують стрічку новин на основі дій користувача, не інформуючи про механізм добору, що викликає занепокоєння у 72% опитаних користувачів (згідно з Pew Research Center). Крім того, зберігання та захист таких чутливих даних залишаються слабким місцем багатьох систем. Дослідження показують, що навіть великі компанії не завжди забезпечують належний рівень шифрування або контроль доступу до користувацької інформації. Наприклад, у 2023 році фіксувалися витоки даних користувачів сервісів штучного інтелекту через уразливості в хмарних сховищах. Це свідчить, що відповідальність за конфіденційність фактично повністю лежить на надавачах послуг, тоді як користувачі мають обмежені можливості контролювати власні дані. У результаті навіть технології,

спрямовані на покращення користувацького досвіду, можуть становити потенційну загрозу приватності.

У підсумку можна сказати, що адаптивні інтерфейси справді здатні зробити цифровий світ більш відкритим і зручним для кожного. Вже сьогодні вони доводять свою ефективність — і це підтверджується не лише словами, а й цифрами, прикладами з життя, показниками залученості користувачів. Вони можуть суттєво покращити якість життя, підвищити рівень самостійності та комфорту для людей з інвалідністю, зменшити цифрову нерівність. Хоча адаптивні інтерфейси вже демонструють вражаючі можливості, їхній повний потенціал ще не розкритий. Щоб вони працювали справді ефективно, потрібно вирішити кілька важливих проблем: зробити системи швидшими й стабільнішими, забезпечити їхню коректну роботу на різних пристроях, а також удосконалити алгоритми, які відповідають за адаптацію під потреби користувача. Окремо варто зосередитись на захисті особистих даних — прозорість і безпека в цьому питанні напряму впливають на рівень довіри до технологій. Лише поєднуючи технічний прогрес із етичними принципами, адаптивні інтерфейси зможуть стати справжнім інструментом цифрової інклюзії — доступним і безпечним для всіх.

1.4 Огляд стандартів доступності та вимог до адаптивних інтерфейсів

Для створення ефективного адаптивного інтерфейсу важливо дотримуватися вимог визначених у міжнародних стандартах та рекомендаціях. На сьогоднішній день існують міжнародні стандарти, такі як ISO 9241 (який охоплює аспекти ергономіки взаємодії людини з системами) який включає частину ISO 9241-171:2008 (яка надає рекомендації щодо забезпечення доступності програмного забезпечення для користувачів з різними фізичними, сенсорними та когнітивними можливостями), ISO/IEC 25010:2023, європейський стандарт EN 301 549 підтримки розвитку найкращих цифрових практик для всіх, американський законодавчий акт ADA (Americans with Disabilities Act), який

регулює доступність цифрових продуктів та середовищ для людей з обмеженими можливостями [26] та рекомендації WCAG (Web Content Accessibility Guidelines), які визначають ключові принципи забезпечення доступності: сприйнятливість, функціональність, зрозумілість і стабільність.

Ключові вимоги для створення адаптивних інтерфейсів для людей з обмеженими можливостями, сформовані на основі стандартів ISO/IEC 25010:2023, ISO 9241-11:2018, ISO 9241-171:2008, EN 301 549, ADA та WCAG 2.1 можна згрупувати по трьом основним категоріям: доступність, зручність використання, здатність до взаємодії. Розглянемо кожну категорію більш детально.

Доступність (Accessibility) цифрових інтерфейсів передбачає створення умов, за яких кожен користувач — незалежно від фізичних, сенсорних чи когнітивних обмежень — може повноцінно користуватися системою. Це означає, що інтерфейс повинен бути зручним, зрозумілим і технічно доступним для всіх. Фізична доступність стосується користувачів із порушеннями моторики. Інтерфейс має дозволяти керування не лише мишею, а й клавіатурою, сенсорним екраном, голосом або навіть рухами очей. Важливо, щоб елементи були достатнього розміру, зручні для натискання та інтуїтивно зрозумілі.

Сенсорна доступність охоплює потреби людей із порушеннями зору або слуху. Для користувачів із вадами зору необхідно забезпечити текстові альтернативи для зображень і відео, використання контрастних кольорів і можливість збільшення шрифту. Для користувачів із порушенням слуху варто передбачити субтитри, візуальні сигнали замість звукових сповіщень і доступність інформації у текстовому форматі.

Когнітивна доступність спрямована на спрощення сприйняття та розуміння інформації. Це досягається завдяки коротким і зрозумілим текстам, логічній структурі, чітким іконкам і підказкам. Також важливо враховувати кольорові поєднання, щоб текст залишався читабельним навіть для людей із порушенням сприйняття кольорів.

Технологічна доступність передбачає стабільну роботу інтерфейсу на різних пристроях — комп’ютерах, планшетах, смартфонах — і сумісність із допоміжними технологіями, такими як брайлеві дисплеї, голосові асистенти чи пристрої альтернативного управління.

Контекстуальна адаптація означає, що система має залишатися зручною за будь-яких умов — при слабкому освітленні, шумі чи повільному інтернет-з’єднанні.

Міжнародний стандарт WCAG (Web Content Accessibility Guidelines) визначає чотири основні принципи доступності. Перший — сприйнятність (Perceivable), який означає, що інформація має бути подана у формах, доступних для сприйняття, наприклад, дублювання звуку текстом. Другий — працездатність (Operable): елементи інтерфейсу повинні бути керованими різними способами — мишею, клавіатурою, голосом або сенсорним екраном. Третій — зрозумілість (Understandable), що передбачає просту, логічну і послідовну структуру інтерфейсу. І нарешті, надійність (Robust) — це здатність інтерфейсу стабільно працювати на різних пристроях і підтримувати допоміжні технології.

Зручність використання (Usability) є однією з головних характеристик будь-якого інтерфейсу. Вона визначає, наскільки легко і швидко користувач може досягати своїх цілей під час взаємодії із системою. У контексті адаптивних інтерфейсів зручність використання має особливе значення, оскільки саме від неї залежить, наскільки система буде інтуїтивною, доступною та комфортною для всіх користувачів, зокрема для людей з обмеженими можливостями.

Інтерфейс має бути зрозумілим навіть для новачків. Кнопки та елементи управління повинні мати логічне розташування й чіткі назви. Для користувачів із порушеннями моторики або зору важливо, щоб спеціальні функції, як-от голосове керування або комбінації клавіш, були простими у використанні.

Висока ефективність означає, що користувач може виконувати завдання з мінімальною кількістю дій. Наприклад, форми можуть автоматично заповнюватися, якщо інформація вже збережена в системі. Такі рішення скорочують час і знижують навантаження на користувача. Інтерфейс має бути

зручним для запам’ятовування — навіть після перерви користувач повинен легко згадати, як ним користуватися. Цьому сприяють логічна структура, передбачуваність і впізнавані іконки.

Окрему увагу варто приділити роботі з помилками. Система повинна не лише попереджати їх появу, а й допомагати користувачу їх виправляти. Повідомлення про помилки мають бути зрозумілими та супроводжуватися підказками. Якщо дані введено неправильно, система повинна або автоматично виправити їх, або чітко пояснити, як це зробити.

Користувач має відчувати комфорт і задоволення від роботи з інтерфейсом. Приємний візуальний дизайн, логічна організація елементів і відсутність зайвих деталей створюють позитивний досвід взаємодії.

Ключові принципи зручності використання мають у своїй основі простоту, передбачуваність, універсальність та адаптивність. Простота означає відсутність надмірних функцій і чітку структуру, що полегшує орієнтацію у системі. Передбачуваність забезпечує стабільну поведінку інтерфейсу — користувач розуміє, що відбудеться після кожної дії. Універсальність означає доступність для всіх, незалежно від фізичних можливостей або досвіду користувача. Адаптивність полягає у здатності системи автоматично підлаштовуватись під індивідуальні потреби, наприклад, змінювати розмір шрифту, контрастність або спосіб навігації.

Здатність до взаємодії (Interaction Capability) — це властивість інтерфейсу, яка визначає, наскільки ефективно, зручно та природно користувач може взаємодіяти із системою. Вона забезпечує не лише функціональність, а й комфорт, інтуїтивність та доступність використання для всіх категорій користувачів — незалежно від їхніх фізичних чи когнітивних особливостей. Такий інтерфейс дозволяє людині легко вводити інформацію, отримувати зрозумілий зворотний зв’язок і змінювати налаштування відповідно до власних потреб.

Одним із головних аспектів здатності до взаємодії є гнучкість управління. Це означає, що інтерфейс має підтримувати кілька способів взаємодії з системою.

Користувач повинен мати можливість обирати той метод, який для нього зручніший — традиційний (за допомогою миші чи клавіатури), альтернативний (голосові команди, жести, сенсорний екран) або асистивний (спеціальні пристрої для людей із порушеннями моторики чи зору).

Не менш важливою є реактивність системи, тобто швидка й зрозуміла реакція інтерфейсу на дії користувача. Людина має відчувати, що її команда виконана — натискання кнопки може супроводжуватись короткою вібрацією, анімацією або звуковим сигналом. Такий зворотний зв’язок створює відчуття контролю й підвищує довіру до системи.

Ще один важливий елемент — контекстна адаптивність. Інтерфейс повинен змінюватися залежно від умов використання. Наприклад, у шумному середовищі система може запропонувати текстовий зворотний зв’язок замість голосового, а в умовах слабкого освітлення — автоматично збільшити контрастність екрана. Це дозволяє зберегти зручність і функціональність у різних ситуаціях.

Важливим компонентом є також можливість персоналізації, коли користувач може налаштовувати інтерфейс під свої індивідуальні потреби — змінювати розмір шрифту, кольорову гаму, мову, розташування елементів або спосіб взаємодії. Це створює відчуття комфорту й підвищує ефективність роботи.

Крім того, інтерфейс повинен бути інтуїтивним — зрозумілим навіть для нових користувачів. Людина має легко здогадуватись, як працює система. Наприклад, значок у формі лупи логічно асоціюється з функцією пошуку, а стрілка — з переходом або поверненням назад. Такі асоціації спрощують навчання і роблять взаємодію природнішою.

Сучасні адаптивні системи повинні також підтримувати складну взаємодію, особливо для користувачів, які застосовують допоміжні технології. Наприклад, люди з порушеннями зору можуть одночасно користуватися екранним читачем і голосовим керуванням, що дозволяє ефективно контролювати систему навіть без зорового контакту з екраном.

Підсумовуючи, можна сказати, що адаптивні інтерфейси, створені відповідно до міжнародних стандартів, таких як ISO, WCAG, EN 301 549 та ADA, мають на меті забезпечення інклюзивності і рівності у доступі до цифрових продуктів для всіх категорій користувачів, включаючи людей із фізичними, сенсорними та когнітивними обмеженнями. Доступність, зручність використання та здатність до взаємодії є трьома основними характеристиками, які забезпечують ефективність адаптивних інтерфейсів. Поєднання цих підходів дає змогу створювати універсальні рішення, які підлаштовуються під індивідуальні особливості користувачів і різні умови використання.

Однак на сучасному етапі розвитку цифрових технологій ці стандарти ще не завжди дотримуються належним чином, а подекуди навіть ігноруються. Це свідчить про те, що питання доступності все ще потребує більшої уваги як з боку розробників, так і з боку організацій, які впроваджують цифрові рішення.

2 РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОЇ АДАПТАЦІЇ ІНТЕРФЕЙСУ ПІД ПОТРЕБИ КОРИСТУВАЧА ІЗ ОБМЕЖЕНИМИ МОЖЛИВОСТЯМИ

2.1 Постановка задачі розробки системи автоматизованої адаптаці

У 2025 році спостерігається позитивна тенденція яка передбачає збільшення частки цифрових продуктів які впроваджують найкращі практики із дотриманням стандартів доступності для людей із обмеженнями. Після початку збройної агресії зі сторони російської федерації питання прискорення даної тенденції є актуальною як ніколи раніше. Проте досі залишається проблема статичності інтерфейсів для взаємодії та врахування потреб однієї категорії обмежень, і часткове або повне ігнорування інших категорій. Інтерфейси вимагають ручних налаштувань, прописування правил спрацювання, або жорстко обмежені вже визначеними правилами і не здатні автоматично підлаштовуватись персонально під кожного користувача. Виникає певна нерівність у доступі до продукту або створює труднощі під час користування.

Розвиток технологій штучного інтелекту який спостерігається за останні роки відкриває нові можливості для подолання бар’єрів, з якими стикаються люди із різного роду обмеженнями. Завдяки інтелектуальним інструментам можна створювати комфортні середовища, що підлаштовуються під персональні потреби користувача — змінюють спосіб подання інформації, форму керування чи навіть голосову взаємодію, а також так би мовити цифрових асистентів які візмуть на себе рутинні дії і допоможуть організувати створення того самого комфортного середовища. Вони дозволять вирівняти умови доступу до інформації, освіти, роботи та соціального життя, роблячи технології справді інклюзивними і доступними для кожного.

Тому постає завдання створення концепції інструменту, із адаптацією алгоритму штучного інтелекту, здатного вчитись на поведінці людини та

динамічно реагувати на її поведінку та адаптувати інтерфейс взаємодії у реальному часі під її персональні потреби.

Огляд існуючих рішень та підходів які були описані в першому розділі даної роботи дозволяє зробити висновок, що існуючі рішення зазвичай спрямовані на адаптацію контенту, або тільки додаються субтитри чи голосовий супровід, але майже не береться до уваги розміщення елементів для взаємодії які можуть викликати певний дискомфорт. Застосунок який може бути підключений на будь яку платформу і буде здатний динамічно та персоналізовано адаптувати елементи керування і відображення для кожного індивідуально – вирішує проблему відсутності універсального механізму. Автоматизована адаптація перетворить користувацький досвід на гнучкий, інтелектуально керований процес.

Мета створюваного браузерного розширення полягає у забезпеченні автоматизованої адаптації інтерфейсу під потреби користувачів із проблемами зору із забезпеченням мінімальної взаємодії з боку людини. Модель ШІ повинна “вчитись” на діях користувача, визначати закономірності його взаємодії, передбачати потенційні труднощі й пропонувати зміни візуальних або функціональних параметрів інтерфейсу без участі оператора. Для досягнення цієї мети передбачається використання алгоритмів кластеризації, технологій машинного навчання таких як рекурентні нейронні мережі, які здатні виявляти і враховувати закономірності поведінки, самонавчатися враховуючи контекст і здатні працювати із послідовностями дій розтягненими у часі. Ручне налаштування параметрів буде потрібне тільки на початковому етапі, етапі навчання системи.

Побудова розширення автоматизованої адаптації передбачає виконання низки дослідницьких і технічних завдань. По-перше, необхідно визначити та систематизувати дані про взаємодію користувачів із обмеженнями зору наприклад дальтонізм який передбачає адаптацію кольорових схем. Кожен тип може частково або повністю мати подібні риси із іншим видом обмежень, і це може ускладнити задачу.

Для визначення категорій обмежень допоможе алгоритм кластеризації, такий як k-means. Визначення категорій за допомогою алгоритму дасть змогу визначити, які параметри інтерфейсу мають найсуттєвіший вплив на зручність користування для кожної категорії, поділивши перед цим їх на кластери. По-друге, слід розробити модель збору даних, що відображатиме поведінкові характеристики користувача (час реакції, кількість натискань, послідовність дій, помилки під час навігації). Зібрані дані формують основу для подальшого аналізу й навчання системи.

Подальший етап полягає у виборі методів машинного навчання, придатних для аналізу користувацької поведінки та прогнозування оптимальних змін в інтерфейсі. У межах цього дослідження передбачається дослідження використання, удосконаленої версії рекурентних нейронних мереж LSTM для врахування часових залежностей у поведінці. Результати аналізу мають формувати набір рекомендацій, що передаються до модуля адаптації. Цей модуль автоматично змінює параметри інтерфейсу згідно з прогнозом — наприклад, збільшує шрифт, змінює кольорову схему чи активує спрощений режим навігації. Для того щоб алгоритм роботи залишався ефективним і міг адаптовуватись, важливо передбачити механізм зворотного зв’язку. Він дасть змогу оцінити, чи покращилась взаємодія після внесених змін, і скоригувати навчання алгоритму на основі отриманих результатів. Так формується цикл самонавчання, який із часом підвищує точність рекомендацій і робить систему більш персоналізованою зменшуючи втручання користувача.

Реалізація моделі дасть змогу продемонструвати, як за допомогою адаптації використання технологій штучного інтелекту можна суттєво підвищити якість цифрової взаємодії, особливо для людей із будь-якими видами обмежень. Така модель, здатна самонавчатись і бути інтегрованою у будь-яку платформу незалежно від середовища чи типу пристрою.

2.2. Концептуальна модель браузерного розширення

Концептуальна модель браузерного розширення для автоматизованої адаптації інтерфейсу описує, з яких частин складається механізм та як її окремі частини взаємодіють між собою, починаючи зі збору даних про взаємодію із інтерфейсом, тривалість, контекст, використовуваний пристрій тощо, формуванням профілю поведінки користувача і закінчуючи своєчасними і доречними зміни в інтерфейсі відповідно до потреб користувача. Її мета — створити безперервний цикл, у якому нейронна мережа спостерігає за діями людини, розуміє їх сенс та формує контекс цих дій, адаптує інтерфейс й постійно вдосконалюється приймаючи зворотний зв’язок, і навчаючись із кожної взаємодії. У цій моделі інструмент не вимагає ручних налаштувань, а підтримує діалог із користувачем. Вона спостерігає за тим, як людина взаємодіє з інтерфейсом, де вона відчуває дискомфорт і в майбутньому прогнозує потреби ще до того, як виникне цей дискомфорт, і вносить корекції так, щоб людина майже не помічала самої логіки адаптації — лише результат у вигляді зручнішого, доступнішого інтерфейсу без дискомфорту.

Архітектура додатку для адаптації елементів керування

В основі архітектури системи лежить побудова взаємопов’язаних модулів, кожен із яких виконує свою функцію — від збору та аналізу даних до адаптації інтерфейсу під конкретного користувача. Така архітектура дозволяє масштабувати кожен модуль окремо, за потреби замінювати алгоритми без переробки всієї системи. Також це дає змогу застосовувати різні патерни під час проектування щоб адаптувати розгортання під різні середовища — від мобільного пристрою до веб-додатка чи настільної програми. Наприклад патерн "Adapter" який дозволяє пристосувати існуючий код або компонент до нового інтерфейсу без зміни вихідного коду. Такий патерн дозволяє спростити інтеграцію системи уже в працююче середовище.

Перший елемент – модуль збору даних, який працює постійно і відповідає за спостереження та вимірювання параметрів взаємодії користувача з

інтерфейсом у реальному часі. Його завдання — фіксувати як явні налаштування користувача, так і приховані сигнали поведінки, характеристики пристрою та контекст середовища, у якому відбувається взаємодія. Модуль виконує роль “сенсорного” шару системи: він фіксує вручну обрані схеми кольорів, розмір шрифту, активовані спеціальні можливості, а також характеристики пристрою — тип екрана, його роздільну здатність, наявність клавіатури, мікрофона, акселерометрів чи підтримку синтезу мовлення.

До контекстних параметрів належать час доби, рівень освітлення, наявність стороннього шуму та стабільність мережі. Усі ці фактори впливають на сприйняття інтерфейсу, тому модель враховує їх під час аналізу. Зібрані сигнали передаються в аналітичний модуль у режимі реального часу, де вони попередньо обробляються — нормалізуються й доповнюються похідними показниками, наприклад «ймовірність порушення кольоросприйняття типу Deuteranopia». Процес збору даних має бути максимально непомітним для користувача, щоб не впливати на швидкість роботи чи стабільність інтерфейсу, але водночас достатньо точним і деталізованим для подальшого аналізу.

Подібні підходи широко використовуються в рекомендаційних системах та різних операційних системах: приміром, екосистема Android чи Apple збирає телеметрію про використання яскравості екрана й час доби, і на основі зібраних даних пропонує автоматичне регулювання; медіасервіси Netflix або YouTube застосовують подібний метод збору й аналізу даних для індивідуалізації контенту, контент підлаштовується під вподобання користувача, механізм спрямований не на контент, а на саму оболонку взаємодії, її органи керування. Робота цього модуля ґрунтується на трьох ключових принципах: енергоефективності, мінімальному втручанні в роботу пристрою та дотриманні принципу «privacy-by-design».

Це означає, що дані збираються лише за згодою користувача, зберігаються локально, і проходять процес анонімізації. Завдяки цьому модуль збору даних забезпечує надійну основу для подальшої аналітики, навчання моделей і адаптації інтерфейсу під конкретні потреби людини.

Збір даних основна частина, так як на отриманих даних ґрунтується подальший аналіз і навчання системи. Важливо щоб процес збору даних відбувався непомітно для користувача та був деталізованим. Це не повинно сповільнювати швидкість взаємодії чи стабільність роботи інтерфейсу.

Наступний компонент модуль обробки даних, який здійснює первинну фільтрацію, легку нормалізацію та анонімізацію інформації за потреби. Він виконує важливу роль у забезпеченні якості інформації що надходить. Цей етап необхідний та можна сказати критично важливий, оскільки саме від точності та достовірності даних залежить ефективність роботи та навчання механізму адаптації. Призначення модуля усунення шумів, технічних помилок, дублювання даних, випадкових викривлень, що можуть виникнути під час зчитування користувацьких дій. Для цього застосовуються методи статичної фільтрації, як ковзне середнє або медіанна фільтрація, які націлені на зберігання стабільності показників.

Далі дані проходять етап нормалізації та масштабування — їхні значення приводяться до єдиного діапазону за допомогою методів Min–Max або z-score стандартизації, що забезпечує коректну взаємодію між різними типами параметрів (наприклад, часом реакції, частотою кліків тощо). Щоб уникнути прогалин або неповноти записів, модуль застосовує інтерполяцію чи заповнення середніми значеннями, щоб відновити відсутні дані зберігаючи при цьому послідовність. Окрім підготовки даних, модуль повинен виконати функції базової безпеки та конфіденційності. Він видаляє персональні ідентифікатори, маркери або будь-які дані, за якими можна встановити особу користувача. Для цього застосовуються методи псевдонімізації, хешування (SHA-256) або диференційної приватності, що гарантують анонімність інформації без втрати її аналітичної цінності.

Завдяки цьому модулю модель ШІ отримує структуровані, очищені і захищені масиви даних, які у подальшому стають основою для аналітики, кластеризації користувачів і навчання моделей штучного інтелекту.

В цьому модулі проводиться і глибша фільтрація, агрегація та узагальнення накопиченої інформації. Модуль виконує функцію глибшого аналізу й узагальнення інформації, яка накопичується.

На першому етапі модуль отримує вже очищені та нормалізовані дані з бази, агрегує їх і створює узагальнені статистичні показники — середні, медіанні, мінімальні та максимальні значення, частоти повторення певних дій, часові тренди тощо. Завдяки цьому можна виявити патерн поведінки користувача, наприклад: як часто він змінює контрастність, яка його швидкість реакції або в які періоди доби взаємодія з інтерфейсом найактивніша.

Другим кроком є виявлення закономірностей і трендів. Для цього застосовуються методи часових рядів, ковзного середнього, автокореляційного аналізу або згладжування експоненційним методом. Такі підходи дозволяють не лише відстежувати поточні зміни у поведінці користувача, а й прогнозувати їхній розвиток у майбутньому. Наприклад, якщо система помічає відповідний патерн, наприклад, що користувач поступово збільшує масштаб інтерфейсу, вона може передбачити потребу в адаптації шрифту чи контрастності наперед.

Окрім статистичного аналізу, модуль періодичної обробки виконує кластеризацію даних, тобто групування користувачів за спільними характеристиками або поведінковими моделями. На цьому етапі застосовується алгоритм K-Means, який дозволяє формувати та групувати профілі юзер. Наприклад, одна група може характеризуватися швидкою реакцією і точними діями, інша — повільним темпом роботи й частими повторними натисканнями. Це дає змогу системі розуміти, як і які адаптації впливають на різні типи абонентів, і вподальшому приймати рішення на основі виявлених закономірностей. На основі профілів користувачів створюються шаблони поведінки для кожного типу обмеження. Ці шаблони будуть застосовані як основа для взаємодії зі споживачем на початковому етапі, і в подальшому профіль буде шляхом донавчання моделі покращуватись і бути точнішим. І цей вдосконалений профіль буде застосований для інших юзерів які за поведінкою відповідають категорії.

Модуль також може включати механізми виявлення аномалій, які сигналізують про нетипову поведінку, збої або помилки у взаємодії. Для цього використовуються статистичні тести або алгоритми машинного навчання, наприклад, Isolation Forest чи DBSCAN. Це дає можливість не лише підвищити точність адаптацій, а й вчасно помічати і реагувати на потенційні проблеми в системі.

Модуль аналізу виконує роль «мозку». Саме тут відбувається робота RNN, яка навчається на даних із сформованого на попередніх етапах профілю й намагається “зрозуміти” логіку поведінки людини. На відміну від простих моделей, RNN працює з послідовностями подій. Це означає, що вона бачить не окремі дії, а цілі ланцюги взаємодій — і може визначити, як одна дія випливає з іншої.

Основна задача нейронної мережі — знайти індивідуальні патерни. Наприклад, як швидко людина реагує на елементи, коли їй легше читати текст, чи змінюється її взаємодія залежно від часу доби. RNN запам’ятовує такі закономірності й, що важливо, може передбачати наступні дії. Якщо відвідувач часто збільшує масштаб або повільніше рухає курсором у темний час доби, нейронна мережа здатна «впізнати» цю тенденцію та передбачити, що інтерфейс краще одразу адаптувати.

У системі використовуються LSTM яка є вдосконаленим типом класичної рекурентної мережі. Вона здатна не просто "бачити" окремі дії, а й запам’ятовувати важливу інформацію з попередніх кроків. Завдяки цьому модель аналізує не лише поточний стан користувача, а й його поведінкову історію: що він робив до цього, з якою швидкістю взаємодіяв, які налаштування змінював, наприклад протягом однієї сесії користування.

Після проходження навчання мережа формує динамічну модель користувача. Це фактично набір прогнозів, що описують, чого він може потребувати в найближчий момент. Наприклад, алгоритм ШІ може передбачити, що людині буде легше працювати з більшим шрифтом, підвищеним контрастом або спрощеною навігацією або якщо темп дій уповільнюється, це може

сигналізувати про втому чи труднощі роботи з інтерфейсом — і модель адаптується відповідно.

Модуль аналізу не залишається незмінним після першого навчання. Він постійно оновлює свої уявлення про персону, реагуючи на нові дії та зміни в умовах, у яких людина працює. Тобто модель буде донавчатись уже під кожного особисто. Інтерфейс поступово підлаштовується, коли користувач змінює свій стиль взаємодії, наприклад вона стала краще бачити через якийсь час, чи рухові здібності відновились, або юзер змінює пристрій або працює в іншому середовищі так з часом вона стає гнучкою.

Завдяки такій гнучкості модуль може діяти наперед. Він визначає моменти, коли користувачеві може знадобитися допомога чи адаптація, ще до того, як той відчує дискомфорт або спробує змінити налаштування вручну. Це робить взаємодію з інтерфейсом плавною й комфортною, особливо для людей, які мають різні види обмежень і потребують додаткової підтримки.

Модуль адаптації інтерфейсу відповідає за те, щоб перетворити результати аналізу на конкретні й зрозумілі зміни в інтерфейсі. Він визначає, які саме елементи потрібно змінити, щоб користувачеві було зручніше працювати. Для цього використовуються заздалегідь визначені правила — що саме має зробити алгоритм в тій чи іншій ситуації. Наприклад, якщо аналіз показує, що відвідувачу потрібна підвищена контрастність, інтерфейс автоматично перемикається на темну тему або вмикає чіткіші контури. Якщо виявлено потребу в моторній підтримці, збільшується розмір клікабельних зон, подовжуються таймаути, з’являються додаткові способи введення тощо. Коли ж потрібно зменшити когнітивне навантаження, додаток спрощує меню, приховує зайві елементи та пропонує покрокові підказки. Саме RNN формує прогноз поточних потреб клієнта, генерує і передає у модуль впорядкований набір показників. В цьому наборі містяться ознаки ймовірності того, що людині потрібні певні зміни інтерфейсу. Дані передаються у вигляді структурованого об’єкта з числовими значеннями, які описують пріоритет кожної потреби.

Модуль стежить за тим, щоб зміни не суперечили одна одній і реагує відповідно до встановлених пріоритетів. Окрім цього, варто звернути увагу, що зміни застосовуються поступово, щоб користувач не відчував різких стрибків в інтерфейсі. Це можуть бути плавні переходи, підказки про те, що відбулося, або можливість швидко скасувати дію, якщо вона виявилась незручною. Подібні механізми вже працюють у багатьох реальних додатках. Наприклад, у програмах для озвучування тексту автоматично читається той елемент, який опинився у фокусі. А на смартфонах клавіатура може сама змінювати розкладку залежно від того, як людина набирає текст. У цьому модулі реалізується схожа ідея — інтерфейс підлаштовується під користувача саме тоді, коли йому це потрібно.

Модуль зворотного зв’язку завершує весь цикл і починає новий. Його головна мета збирати як прямі реакції людини, так і непрямі сигнали, які показують, чи справді зміни були корисними. (наприклад, короткі запити «Чи стало зручніше?» із можливістю швидкої відповіді, або скорочення часу на виконання операцій, зменшення кількості помилок, зниження частоти ручних корекцій інтерфейсу).

Ці зібрані дані перетворюються на прості метрики які використовуються для оновлення профілю та перевірки коректності спрацювання попередніх прогнозів. Щоб не перевантажувати людину, додаток мінімізує кількість запитів і віддає перевагу вимірюванню результатів. Якщо ж рівень невизначеності високий, модуль може запропонувати декілька варіантів змін, людина обере найкращий для неї. Варто врахувати, що такі запити будуть створюватись тільки на етапі початку роботи. Чим довше людина буде користуватись системою тим більш персоналізованішими будуть зміни.

База даних не виділена як окремий модуль, але є важливою частиною всієї системи. Саме тут зберігається вся інформація про взаємодію користувача, результати аналізу, проміжні обчислення та оновлені профілі. Без цього компоненту механізм просто не зміг би збирати дані та працювати з цими накопиченими даними, навчатися на них і будувати прогнозовані моделі. База забезпечує цілісність процесу: модулі можуть працювати незалежно, але всі вони

звертаються до одного спільного джерела інформації, це гарантує узгодженість і доступність потрібної інформації у потрібний момент. Так як і інші модулі, база даних не прив’язана до конкретної технології і може бути використана будь-яка актуальна база даних.

Архітектура браузерного розширення для адаптації інтерфейсів для людей із проблемами зору представлена на рисунку 2.1.

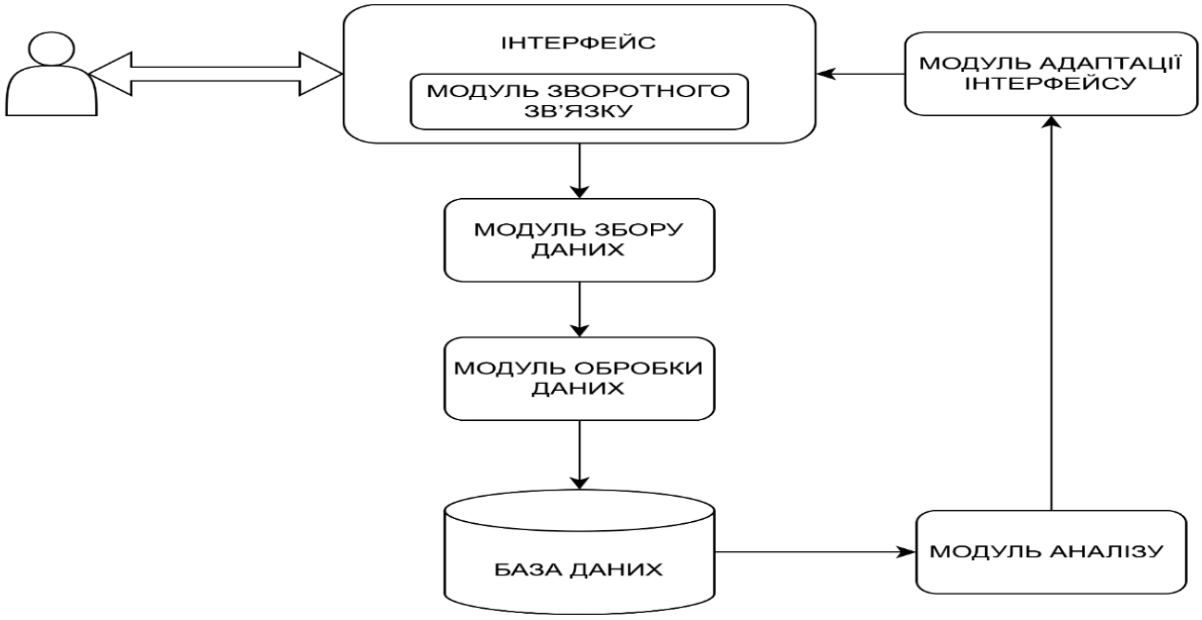


Рис. 2.1 Архітектура браузерного розширення для адаптації інтерфейсів для людей із проблемами зору.

2.3. Опис використаних моделей кластеризації і алгоритму штучного інтелекту

Простих, раз і назавжди зафіксованих правил, які традиційно застосовують в інтерфейсах доступності, для поставленої задачі вже недостатньо. Виникає питання чому? Одна й та сама людина поводиться по-іншому вранці й пізно ввечері, на смартфоні й на великому моніторі, у тихому кабінеті й у шумному транспорті. Для людей з інвалідністю ця різниця ще відчутніша: дрібний шрифт може бути прийнятним у добре освітленому офісі, але перетворюється на проблему в напівтемній кімнаті; рухи миші стають менш точними під час втоми;

складне багаторівневе меню перевантажує когось, хто зазвичай добре орієнтується в структурі. Тому концепція системи автоматизованої адаптації інтерфейсу спирається не на один алгоритм, а на багаторівневу обробку сигналів (своєрідний конвеєр). На одному рівні потрібно розкласти різні сценарії взаємодії на типи які будуть стійкими. Щоб можна було швидко зрозуміти сценарій поведінки. На другому — відстежити, як змінюються дії користувача в часі, щоб ці дані допомогли передбачати ці дії. На третьому — дані перетворити на прогнози і ухвалити конкретне, бажано пояснюване рішення: що саме змінити просто зараз, а чого краще не чіпати.

У запропонованій архітектурі ці ролі розподілені між трьома сімействами методів: K-Means, рекурентними нейронною мережею LSTM та деревоподібними моделями ухвалення рішень. Разом вони утворюють певний «конвеєр»: K-Means або подібні алгоритми кластеризації, групують схожі сесії в кластери й дає компактний опис типового сценарію; RNN працює вже з часовою послідовністю дій і прогнозує, яка адаптація найімовірніше знадобиться в найближчий момент враховуючи контекст; дерева рішень перетворюють отримані ймовірності та кластерні ознаки на конкретні зміни інтерфейсу, які будуть зрозумілими користувачу. Один блок відповідає на запитання «хто перед нами», другий — «що, ймовірно, відбудеться далі», третій — «як саме зараз змінити інтерфейс».

На першому етапі повинен структуруватись набір патернів поведінок (поведінка може змінюватись залежно від контексту). Для цього використовується алгоритм K-Means — класичний метод кластеризації, який намагається знайти такі центри кластерів, щоб точки всередині кожного кластера були максимально схожими між собою. Математично задача виглядає як мінімізація функції:

$$J = \sum_{i=1}^{k} \sum_{x_j \in C_i} ||x_j - \mu_i||^2 \quad (2.1)$$

де k— кількість кластерів,

C_i— і-тий кластер,

x_j— вектор ознак j-тої сесії,

μ_i— центр цього кластера.

В одному такому векторі поєднуються частота ручних збільшень шрифту, кількість перемикань тем протягом сесії, рівень освітлення, шумове тло, тип пристрою. Усі ці величини попередньо нормалізуються, щоб, наприклад, високий рівень освітлення не впливав на результат більше, ніж часті промахи. На виході K-Means дає нам індекс найближчого кластера.

Кількість кластерів k не задається навмання. Її підбирають, орієнтуючись на «метод ліктя» (elbow method) для функції $J(k)$, та на те, як ці групи можна змістовно інтерпретувати. Наприклад, одна група це темна тема у поєднанні з великим шрифтом, інша — часті промахи і повільний рух. На виході кластеризатор для кожної сесії повертає індекс найближчого кластера та відстань до його центру; ці два числа далі використовуються як стислий «паспорт» поведінки. В практиці доступності такі кластери зазвичай непогано узгоджуються: одна група відповідає користувачам, які регулярно збільшують шрифт і перемикаються на темні теми ввечері; інша — тим, хто часто промахується по маленьких елементах і рухається повільно; ще одна — сценаріям, де людина губиться в глибокому меню та неодноразово повертається назад.

Однією з сильних сторін K-Means це здатність оновлювати центроїди поступово, без необхідності повного перерахунку моделі щоразу, коли надходять нові дані. Завдяки цьому алгоритм легко інтегрується у роботи в реальному часі, де нейронна мережа постійно отримує нову інформацію про користувача. Така властивість робить кластеризатор зручним для побудови динамічних адаптивних інтерфейсів, у яких важливо швидко реагувати на зміни та не проводити зайвих обчислень. Але існують і слабкі сторони які варто врахувати. Він погано працює

у випадках, коли кластери мають різну форму, відрізняються за густиною або не являються близькими до сферичної структури, також він чутливий до масштабу окремих ознак: параметри з великими значеннями можуть непропорційно впливати на результат, якщо дані не були попередньо вирівняні.

Ефективність кластеризації значною мірою залежить від правильно обраної кількості кластерів, тоді як оптимальне значення k не завжди очевидне і може змінюватися з часом. Але якщо дані проходять нормалізацію, тоді це зменшує вплив різних масштабів ознак. Кількість кластерів періодично переоцінюється, щоб модель залишалася релевантною до змін поведінки абонента. Додатково оцінюється якість отриманого розбиття, орієнтуючись на фактичні результати адаптації інтерфейсу, що дозволяє своєчасно реагувати на деградацію моделі та коригувати параметри кластеризації.

Кластеризація відповідає на запитання, «до якого сценарію подібна поведінка користувача», але не “вміє” прогнозувати. Для цього потрібна модель, що працює з послідовностями даних, рекурентні нейронні мережі.

На відміну від звичайних нейронних мереж, що обробляють кожен вхід незалежно, RNN «пам’ятають» попередні стани і завдяки цьому враховують контекст. Саме ця властивість і робить їх цінними в продуктах, де поведінка юзера змінюється в часі. Прямі мережі передають дані в одному напрямку від умовного входу до виходу і погано передбачають наступний крок, тому що не враховує вплив інших даних на часовому проміжку. RNN у свою чергу на кожному кроці повертають оброблену інформацію назад у мережу за допомогою циклу і при наступному прийнятті рішення враховує як поточні дані так і попередньо вивчені.

Цей процес можна пояснити на прикладі читання реченя де ви намагаєтесь вгадати наступне слово, але ви не опираєтесь тільки на поточне слово яке бачите, ви враховуєте і пам’ятаєте ще попередні. От і рекурентні мережі працюють за схожим принципом, зберігаючи інформацію про попередні кроки і передають вихід одного кроку як вхід до наступного. Таким чином враховуються попередні слова для того щоб спрогнозувати яке буде наступне.

Схематично різниця між видами мереж представлена на рисунку 2.2.

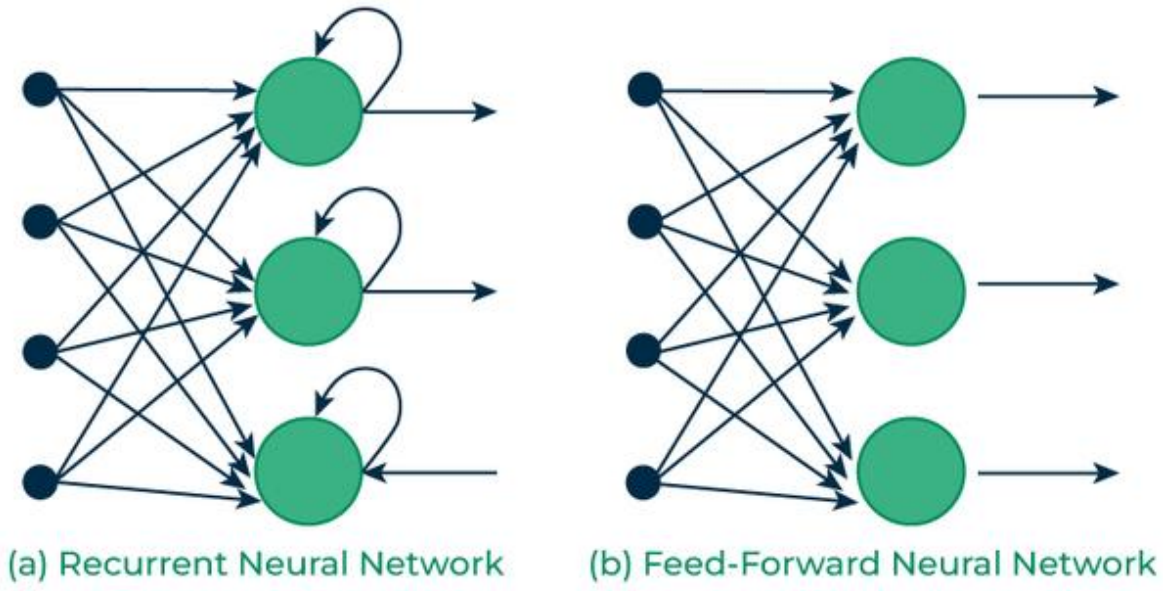


Рис. 2.2 Схема порівняння рекурентних і прямих нейронних мереж.

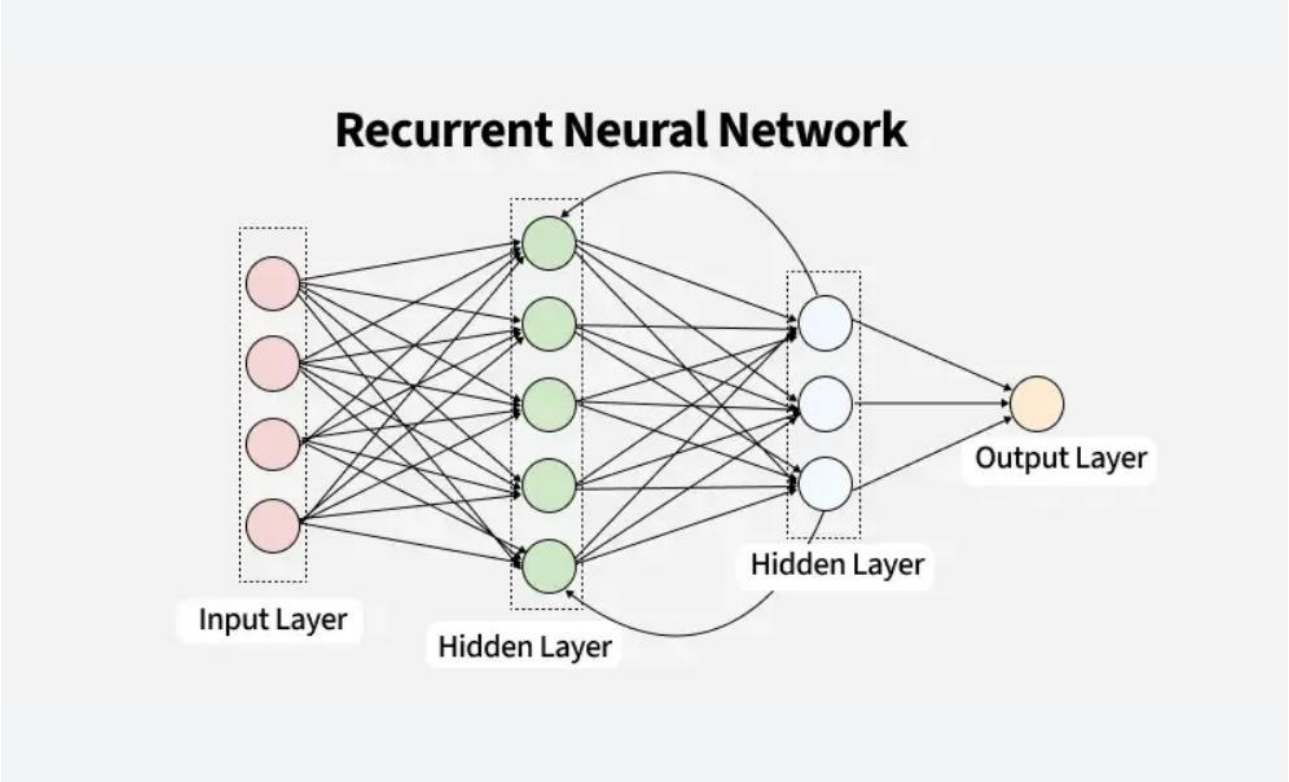


Рис. 2.3 Класична схема рекурентної нейронної мережі

Розглянемо роботу RNN на прикладі предмету дослідження. Взаємодія користувача з інтерфейсом природно описується не одним вектором, а послідовністю станів. Нехай $X = [x_1, x_2, \dots, x_T]$ — послідовність коротких «кадрів» по кілька секунд, де кожен вектор x_t містить поточний розмір шрифту, кількість помилок за останній проміжок, швидкість прокручування, активну тему, оцінений контраст щодо освітлення, час реакції, тип активності (читання, введення, навігація). Основна ідея RNN полягає у введенні прихованого стану h_t, який оновлюється на кожному кроці часу залежно від попереднього стану і поточного вхідного сигналу. Так мережа формує внутрішню пам'ять, що відображає історію взаємодії.

Математично прихований стан оновлюється за формулою:

$$h_t = f(\mathrm{U}X_t + Wh_{t-1} + b_h) \tag{2.2}$$

де h_t — прихований стан на етапі часу t,

x_t — це поточний вхідний вектор (поточний кадр поведінки користувача),

U — матриця ваг, які застосовуються до поточного вхідного сигналу x_t. Ця матриця визначає, наскільки важлива поточна інформація для оновлення пам'яті мережі,

W— матриця ваг, яка застосовуються до попереднього прихованого стану h_{t-1} тобто до пам’яті мережі,

b_h — вектор зміщення для прихованого шару,

а f — нелінійною функцією активації tanh.

Вихідний вектор, що описує прогнозовані потреби, отримується як

$$h_t=\sigma(W_h h_{t-1} - W_x x_t - b_h) \tag{2.3}$$

h_t- вектор внутрішнього стану моделі на момент t,

W_h, W_x- матриці ваг зв’язків,

b_h- вектор зміщень,

σ- нелінійна активаційна функція сигмоїда, нелінійна функція активації може бути softmax або сигмоїдою, залежно від формулювання задачі.

У нашому випадку вектор y_t зручно інтерпретувати як набір імовірностей

$$y_t=[p_{\text{font}}, p_{\text{contrast}}, p_{\text{color}}, p_{\text{cognitive}}] \quad (2.4)$$

де, p_{font} показує, наскільки ймовірно, що в найближчий час шрифт доведеться збільшити,

p_{contrast}— потребу у підвищеній контрастності,

p_{color}— у зміні кольору шрифта,

$p_{\text{cognitive}}$— у спрощенні інтерфейсу та покрокових підказках.

На практиці проста RNN швидко стикається з проблемою зникнення градієнту при навчанні на довгих послідовностях. Через те що градієнт повинен передаватись назад у часі щоб оновлювати ваги на всіх попередніх кроках, один і той самий коефіцієнт перемножується сам на себе багато разів і швидко прямує до нуля, а через десятки і сотні перемножень взагалі зникає. Як наслідок цього мережа не може ефективно працювати із довгими залежностями. Варто також сказати за обернену ситуацію – вибух градієнта. Градієнт може стати занадто великим що призведе до проблем зі стабільністю роботи і ускладнює коректне навчання моделі бо оновлення стають непередбачуваними. Тому щоб уникнути проблем із градієнтом використовуються спеціально створену — LSTM мережа.

Що таке LSTM?

LSTM-мережі це вдосконалена версія звичайних рекурентних нейромереж, оскільки вони також працюють із контекстом попередніх кроків у послідовності та усувають проблему зникнення і вибуху градієнта.

Їх ключова відмінність, у порівнянні з традиційною RNN, яка використовує один прихований стан, що проходить у часі, що LSTM мають спеціальну комірку пам’яті, що дає змогу краще “пам’ятати” важливу інформацію з минулого, і може довше зберігати цю інформацію в пам’яті. Архітектура LSTM представлена на рисунку 2.4.

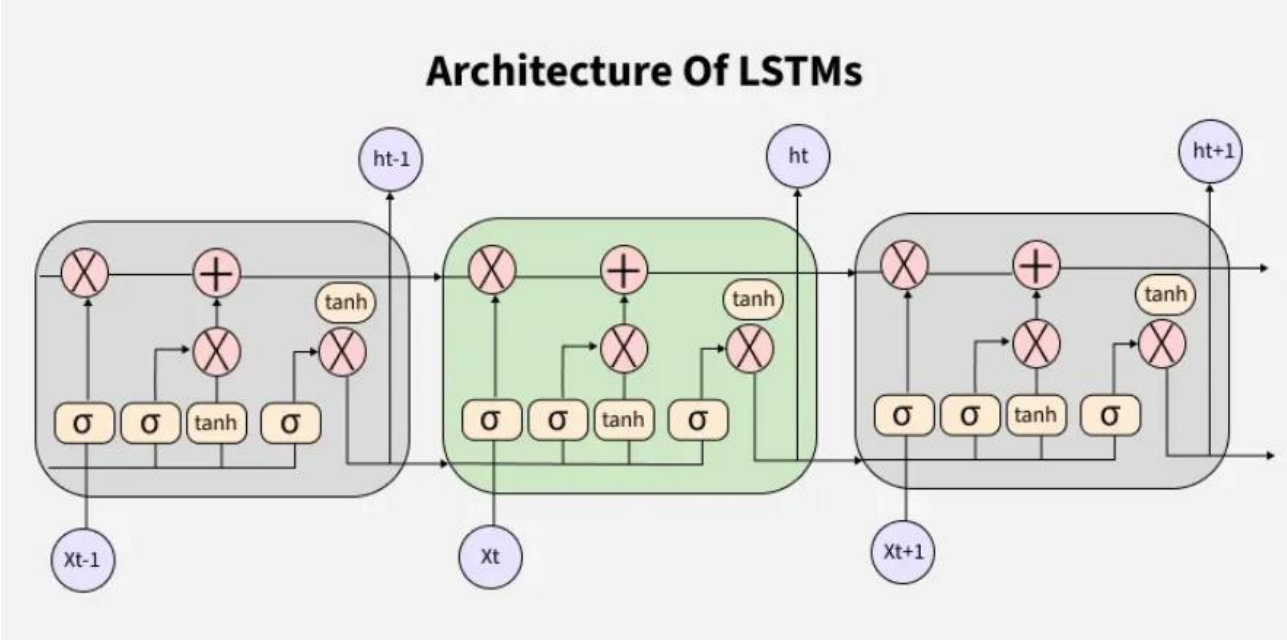


Рис. 2.4 Архітектра мереж LSTM (Довготривалої короткочасної пам'яті)

У центрі архітектури LSTM яка має спеціальний «механізм пам’яті», стан комірки C_t— довготривала пам’ять, що проходить крізь кожен елемент мережі. Цей стан може змінюватись спеціальними керуючими механізмами які називають воротами (gates). Їхня роль полягає у контролі, яку інформацію потрібно “запам’ятати”, яку “забути” та що передати далі. Ворота працюють на основі функції активації сигмоїди, яка повертає значення від 0 до 1 і тим самим вирішуючи, яку частину інформації пропустити далі, а яку відкинути і не пропускати.

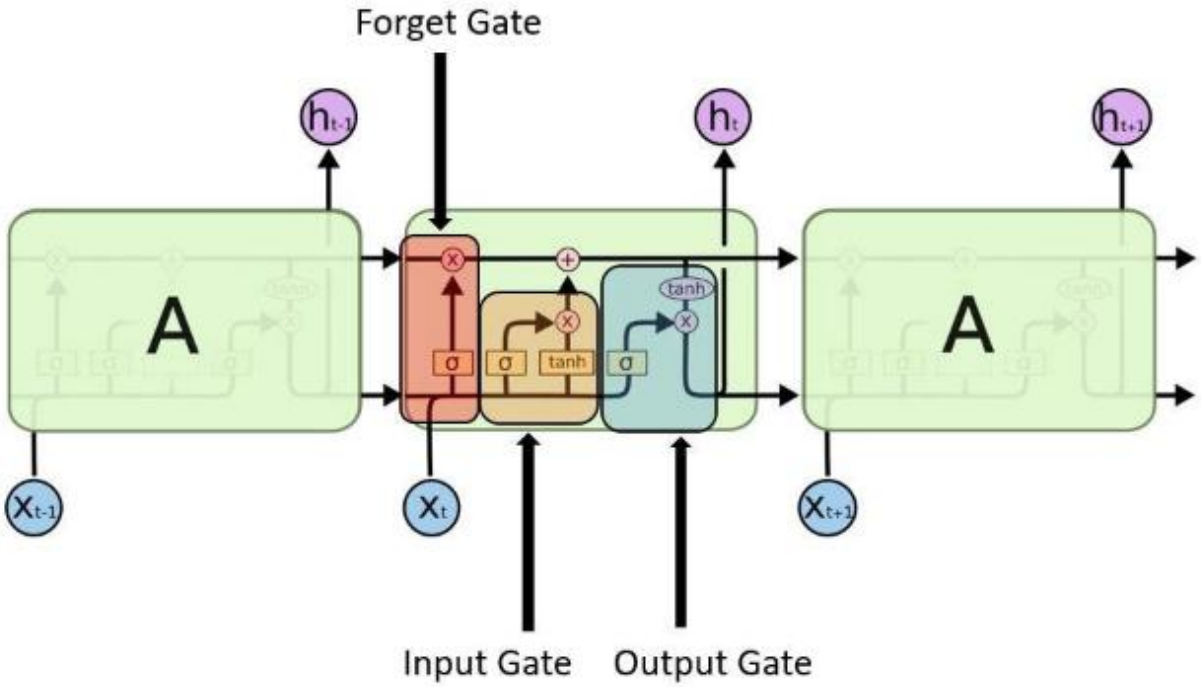


Рис. 2.5. Модель LSTM

Ворота забування (forget gate) це крок на якому вирішують, яку частину минулого досвіду можна відкинути із елемента через його неактуальність. Сигмоїда “дивиться” у попередній стан h_{t-1} та вхідний стан x_t і вирішує кого пропускати присвоївши значення 1, а кого забути присвоївши значення близьке до 0. Наприклад у контексті поведінки користувача, якщо його поведінка вдень відрізняється від поведінки вночі, то перед переходом до аналізу поведінки вдень LSTM “забуде” шаблон дій вночі. Математична формула обрахунку воріт забуття представлена нижче.

$$f_t = \sigma\left(W_{xf}^T x_t + W_{hf}^T h_{t-1} + b_f\right) \tag{2.5}$$

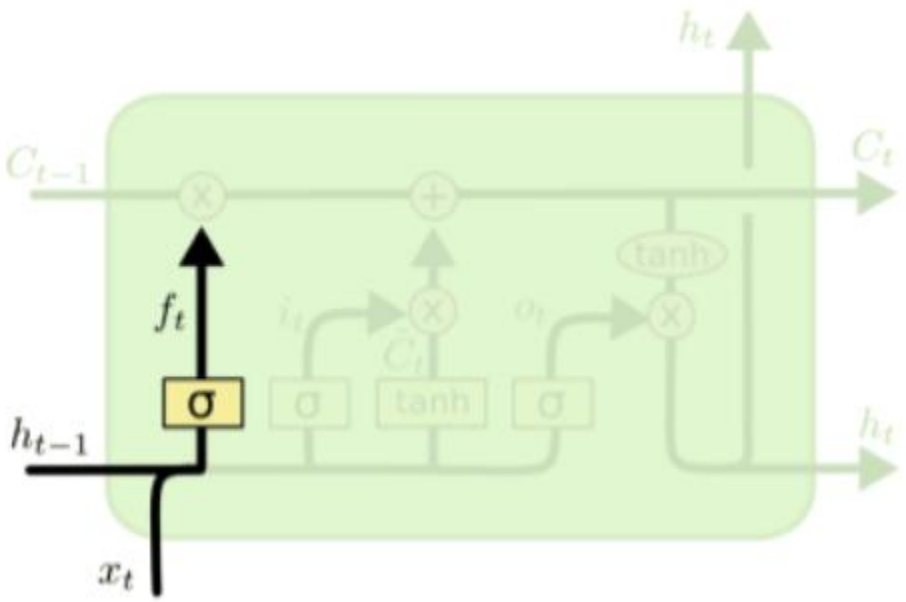


Рис. 2.6 Схема роботи воріт забуття (forget gate)

Ворота запам’ятовування (input gate) або вхідні ворота визначають, що важливо і треба зберегти в пам’яті, додати до стану комірки. Функція tanh активує ваги значенням визначаючи їх важливість в діапазоні від -1 до 1.

Після цього LSTM комбінує стару й нову інформацію, визначаючи, що необхідно оновити в комірці. Дану комбінацію можна представити як процес оновлення стану елементу представивши у формулах нижче.

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \tag{2.6}$$

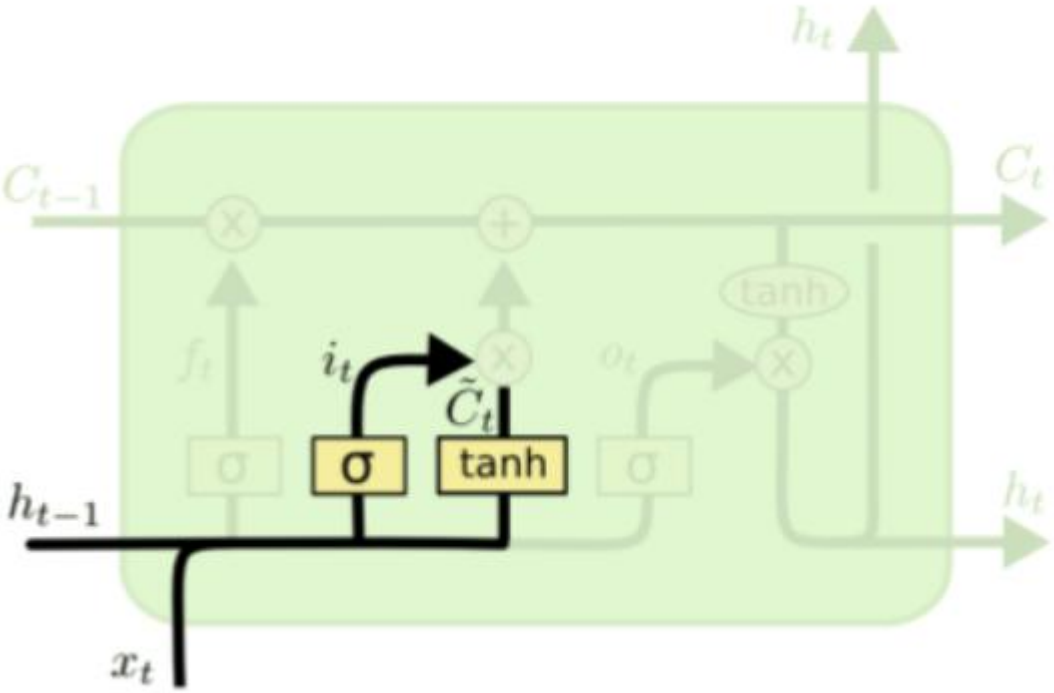


Рис. 2.5 Ворота запам’ятовування (input gate)

Наступним етапом відбувається оновлення стану комірки. В результаті частина пам’яті залишається, а частина змінюється новими значеннями. Так механізм плавно оновлює довготривалі залежності. Попередній стан C_{t-1} потрібно оновити на новий C_t, це вирішено уже в попередньому кроці, що оновити. Здійснюється множення C_{t-1} тобто попередній стан на f_t тим самим забуваючи інформацію яку вирішено було забути у нейроні звідки прийшов f_t. Перемноживши значення i_t та $\tilde{C}_t$ отримуємо нові значення.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tanh(W_{xc}^T x_t + W_{hc}^T h_{t-1} + b_c) \qquad (2.7)$$

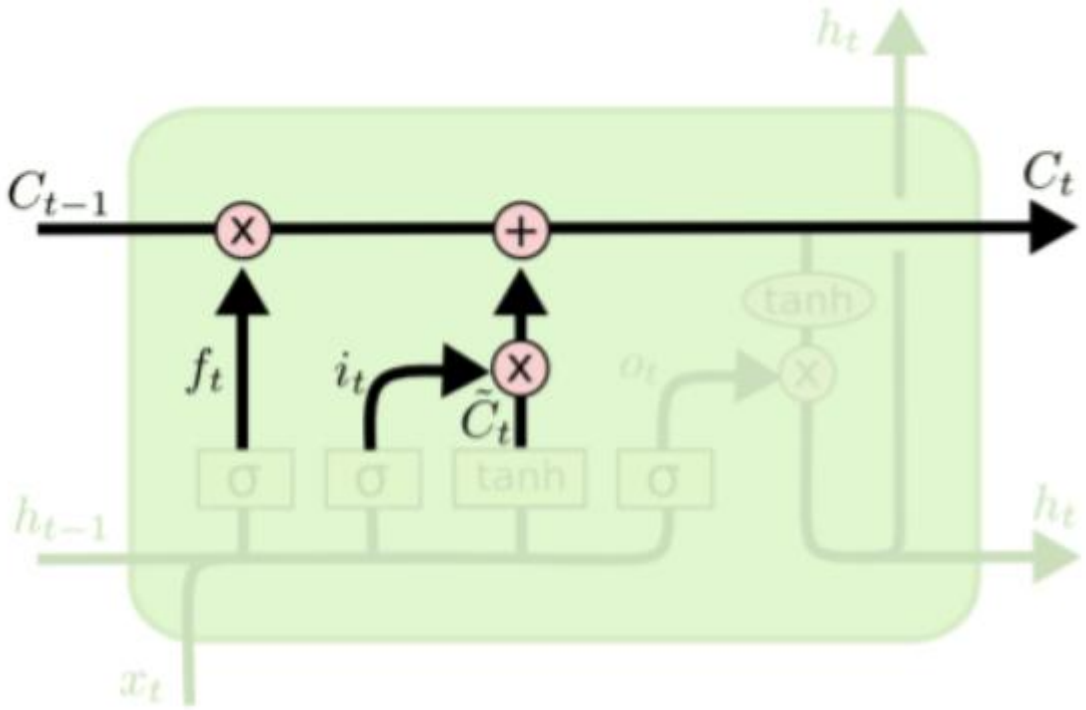


Рис. 2.6 Оновлення стану елемента

Останній крок це вирішити яку інформацію передати як вихідний стан іншими словами, що саме із збереженої інформації буде використано далі. Цей вихід називають (output gate).Першим запускається сигмоїдний нейронний шар, який і визначає яку частину стану комірки подати на вихід. Далі стан комірки пропускається через tanh та перемножуються із вектором стану поданим з вихідного затвору, пропустивши через ворота виходу значення яке вирішено сигмоїдою. Математичні формули процесу представлені нижче.

$$\begin{aligned} o_t &= \sigma(W_{xo}^T x_t + W_{ho}^T h_{t-1} + b_o) \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{2.8}$$

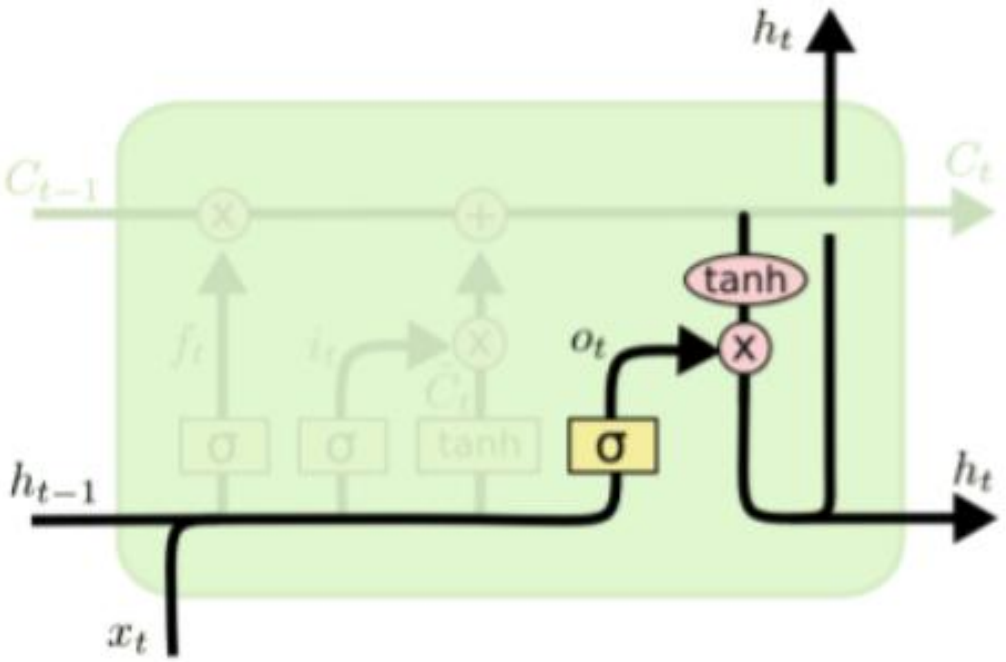


Рис. 2.7 Вихідні ворота (output gate)

Навчання RNN відбувається в режимі з учителем. Нейронна мережа накопичує історії сесій, де відомо, коли користувач сам змінював налаштування (збільшував шрифт, вмикав високий контраст, переходив у спрощений режим) і які автоматичні адаптації реально покращили об’єктивні показники — скоротили час до цільової дії, зменшили кількість помилок або ручних втручань.

Ці моменти виступають цільовими подіями, на які підганяються параметри мережі. Перевага такого підходу в тому, що мережа бачить не окремий «кадр», а цілий ланцюжок: повільне читання, зростання числа промахів, ручне збільшення шрифту — і вчиться випереджати цей момент.

Але одночасно існує ризик перетренування на частих сценаріях і чутливість до зміни розподілів. Тому використовуються регуляризація, рання зупинка, періодична перевірка на відкладених наборах і простий «детектор зсуву»: якщо нові дані занадто відрізняються від тих, на яких мережа навчалася, її прогнози тимчасово вважаються менш надійними, і систему частіше підстраховують консервативні правила.

Навіть маючи хороший кластер та якісний вектор прогнозів y_t, цього недостатньо, щоб зміни в інтерфейсі застосувались автоматично. Замість жорстко прописаних правил на кшталт «if–else» використовуються дерева, а інколи — навіть ансамбль дерев, наприклад випадковий ліс.

На вхід цього блоку, окрім прогнозів RNN, потрапляє інформація про кластерну приналежність та контекст: чи були нещодавні ручні відміни автоматичних змін, що зараз робить споживач — читає довгий текст, заповнює форму чи просто перегортає стрічку. Кожен внутрішній вузол дерева перевіряє щось дуже конкретне і просте, наприклад «чи перевищує p_{font} поріг 0,8» або «чи збігається активний кластер з умовно вечірнім сценарієм із низьким освітленням». На листку знаходиться конкретний набір дій: плавно збільшити шрифт на один крок, увімкнути висококонтрастну тему, розширити мінімальний розмір інтерактивних зон, прибрати другорядні елементи з екрана й додати короткі підказки для поточного кроку.

Таке дерево можна уявити як гнучка і «розумніша» версію набору правил. Воно зберігає простоту розуміння де легко відслідкувати причина-наслідок: ланцюжок перевірок, який привів до рішення, можна записати в лог і за потреби використати для аналізу. Якщо включено висококонтрастний режим, завжди можна пояснити: це сталося тому, що модуль збору даних побачив високі значення p_{contrast}, вечірній час доби, тьмяне освітлення і кілька попередніх ручних перемикань теми. Серед обмежень деревоподібних методів – здатність простого дерева «підлаштуватися» під випадкові особливості вибірки й складно описати взаємозалежні умови, коли зміна одного параметра можлива лише за певної комбінації інших. Ці ризики можна компенсувати обмеженням глибини дерева, регулярним перенавчанням і введенням додаткових перевірок для критичних дій.

Як усе це працює разом, легше побачити на простому сценарії. Нехай юзер вечорами часто збільшує шрифт і перемикає тему на темну, працюючи з ноутбука в напівтемній кімнаті. K-Means стабільно відносить такі сесії до одного й того самого кластера, центр якого описує «вечірній візуальний сценарій». Після кількох подібних вечорів RNN починає щоразу на старті сесії видавати високі

значення p_{font}і p_{contrast}, щойно освітлення затемнюється і користувач відкриває довгий текст. Дерево, отримавши кластер, прогнози та контекст, ухвалює рішення збільшити шрифт і увімкнути підвищений контраст ще до того, як людина знову зайде в налаштування. Якщо після цього час до читання скорочується, а ручні корекції відсутні, модуль зворотного зв'язку фіксує це як вдалі адаптації, і наступного разу така зміна застосовується швидше і у розрахунках мережі вона ніби підкріплюється.

Або може бути й протилежна ситуація. Наприклад представимо, алгоритм вирішила, що потрібна зміна елементів взаємодії, збільшила кнопки, а користувачу це завадило: він одразу повернув усе назад, час до виконання завдань збільшився. Така адаптація позначається як невдала і знову відбулося підкріплення від учителя. Під час наступних сесій той самий кластер і подібні прогнози RNN уже не дадуть автоматично такої ж комбінації зміни. Дерево або коригується під час перенавчання, або вводить додаткову умову — вимагати підтвердження перед повторним увімкненням цього режиму, щоб уникнути хибного спрацювання.

Поєднавши три різні типи моделей, можна зробити щоб сильні сторони кожної підсилювали інші, а слабкі взаємно нівелювати. K-Means забезпечує карту поведінкових сценаріїв і дозволяє швидко орієнтуватися, яка модель поведінки. Рекурентні мережі додають часовий вимір і працюють не лише з поточним моментом, але і враховують контекст у декількох попередніх кроках чи періодів взаємодії. Деревоподібні методи закривають частину перетворення числових прогнозів на конкретні зміни інтерфейсу з урахуванням обмежень, стандартів доступності, пріоритетів і реакції користувача. Саме така гібридна архітектура, як цілісна комбінація й здатна забезпечити персоналізовану та автоматизовану адаптацію елементів взаємодії із інтерфейсом, що зможе працювати не лише в лабораторних умовах, а з реальними людьми, реальними пристроями і постійно змінним цифровим середовищем.

2.4 Аналіз обмежень сучасних браузерних розширень і переваги адаптивної системи на основі RNN

Браузерні розширення, якщо дослідити їх детальніше, які допомагають людям із порушенням зору, можна побачити одну спільну рису: вони налаштовуються користувачем вручну. Користувач заходить на сайт наприклад на новинний сайт, і постійно змушений включати темну тему, збільшувати шрифт або налаштовувати собі комфортний контраст. Навіть якщо розширення налаштовується один раз за сесію, при наступному використанні все доведеться починати заново.

Dark Reader, Eye-Able, Valmiki чи Color Enhancer браузерні розширення які допомагають людям із порушенням зору, вони працюють приблизно наступним чином – користувач вручну виконав дію як результат розширення змінило інтерфейс. Для людини зі слабким зором це створює додатковий дискомфорт під час користування. Це навіть змушує відмовлятись користуватись браузером через негативний досвід який щоразу повторюється. Наприклад, користувач із центральною дегенерацією сітківки кожного разу збільшує шрифт у 2–3 рази, тому що для нього текст просто зливається. Інша людина через дальтонізм щоразу вмикає висококонтрастний режим, чи міняє кольори, щоб відрізнити кнопки та посилання. Також варто зауважити що людина по різному сприймає кольори навіть у різну пору доби.

Існуючі на даний момент інструменти часто охоплюють лише частину вимог WCAG 2.2, інколи тільки рівня A або AA, рідкі випадки повного або часткового виконання вимог рівня AAA. Наприклад, деякі розширення збільшують шрифт, але не покращують навігацію клавіатурою. Інші можуть додавати високий контраст, але не прибирають мерехтливі анімації. Або мати чіткі профілі контрасту і не давати змоги налаштувати контраст комфортний персонально. Через це користувачу потрібно комбінувати декілька інструментів, один який гарно виставляє контраст, інший змінює розмір шрифта без

руйнування верстки або налаштовувати кожен сайт окремо, бо після перезавантаження чи переходу “магія” може зникнути.

Розроблена та запропонована система браузерного розширення працює зовсім інакше. Вона повністю відповідає всім вимогам WCAG 2.2 рівня AAA, тобто гарантує максимально можливий рівень доступності згідно вимог для людей із порушенням зору.

Це означає, що користувач може чутливо налаштувати комфортний контраст який підходить йому, зручний розмір шрифту який можна змінити під себе або відповідно до сторінки на якій він знаходиться. Розширення зможе запропонувати зміну кольорової схеми яка буде безпечна для окремих порушень зору. Зручну та повністю доступну навігацію згідно вимог, а саме користувач повинен мати доступ керувати із клавіатури без використання миші. Елементи при масштабуванні залишаються доступні, відсутній горизонтальний скролінг, а також відсутність небажані мерехтіння чи небезпечні ефекти. І все це — на будь-якому сайті, навіть якщо сам сайт взагалі не має відповідності навіть мінімальним стандартам доступності доступності.

Центральне ядро і основна перевага системи – інтеграція рекурентної нейронної мережі завдання якої полягає у прогнозуванні та адаптації інтерфейсу із мінімальним втручанням людини. Найчастіше у інструментах які використовуються людьми із порушенням зору ШІ використовують для озвучування тексту чи зображень, створення субтитрів чи голосового управління. Рекурентні мережі в основному застосовуються для прогнозування контенту який буде показуватись для користувача. Наприклад Netflix який підбирає фільми та серіали на основі ваших попередніх переглянутих, або вподобаних. Youtube та Facebook аналізує вашу поведінку, враховує тип та тематику переглянутих вами відео і на основі цих даних пропонує відповідний для вас контент. Але RNN може аналізувати послідовність і дій людини під час користування — так, ніби як людина формує звичку. Постійно одні і ті самі дії формують звичку аналогічно і нейронна мережа запам’ятовує і формує свій контекст.

Для кращого розуміння наведемо декілька прикладів.

Приклад перший, якщо користувач протягом тижня кожен вечір вмикав темну тему, додаток це запам’ятовує і через деякий час буде вмикати її автоматично.

Приклад другий якщо користувач завжди збільшує шрифт до 165% на новинних чи інфо порталах, то на наступному схожому чи подібному сайті додаток зробить те саме замість людини.

Приклад 3 користувач має дальтонізм і постійно змушений регулювати і змінювати кольори, механізм після вивчення зможе сама підбрати потрібну колірну схему для уникнення дискомфорту.

Виходить так, що нейронна мережа стає зручним інструментом який допомагає автоматизувати рутинні дії, зайшов і користуєшся. Менше дій, більше часу на користування. Інтерфейс більше не вимагає уваги людини, зусилля по налаштуванню бере на себе інструмент – браузерне розширення.

Персоналізований “асистент” який підлаштувався і знає всі звички юзера. Це нагадує ситуацію з сучасними смартфонами: коли телефон розуміє, що ввечері вам зручніше тепле світло, він вмикає його автоматично.

На відміну від інструментів які реагують лише на одну дію чи працюють згідно заданих жорстких правил, RNN будує динамічний профіль поведінки. Існує вірогідність що людина яка використовує розширення, сьогодні має поганий зір і кольоросприйняття, але після курсу терапії чи хірургічного лікування стала краще бачити, відповідно її поведінка під час взаємодії змінюється. Тому виходить так, що чим довше людина користується додатком, тим точніше інструмент визначає її потреби.

Традиційними “ручними” алгоритмами це зробити неможливо оскільки всі зміни постійно потрібно додавати, переписувати існуючі тощо. Але модель, яка вміє працювати з послідовностями дій, здатна виявити закономірності та на основі них підлаштовуватись.

Ще одна важлива перевага — архітекткра програми універсальна. Ця універсальність не залежить від того, чи відповідає сайт стандартам доступності чи дотримується своїх стандартів. Навіть якщо сайт зроблений погано, чи цей

сайт зроблений ще до будь якого роду стандартів, без будь якої адаптивності та WCAG, розширення компенсує це і створить середовище для коректного відображення, яке буде зручне для конкретної людини.

У результаті що маємо, користувач не витрачає час прибравши рутинні дії з ручного налаштування, не перевантажує свій зір діями в непристосованому середовищі і отримує інтерфейс, який персоналізований під нього, навіть без взаємодії з його боку.

Це робить розширення не просто інструментом доступності, а в якійсь мірі особистим помічником, який запам’ятовує, аналізує і прогнозує тим самим допомагаючи людині бачити, читати та взаємодіяти з меншим зусиллям і більшою впевненістю.

3 МОДЕЛЮВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ СИСТЕМИ АВТОМАТИЗОВАНОЇ АДАПТАЦІЇ ІНТЕРФЕЙСУ

3.1. Опис використаних програмних засобів

Розроблення браузерного розширення автоматизованої адаптації інтерфейсу вимагає використання низки сучасних програмних засобів, технологій та бібліотек, кожна з яких виконує своє функціональне завдання та забезпечує коректну взаємодію між клієнтською і серверною частинами. Побудова розширення здійснювалася з урахуванням вимог масштабованості, продуктивності та можливості подальшого навчання моделей на основі даних реальних користувацьких сесій. Тому програмні засоби обиралися таким чином, щоб забезпечити зручну інтеграцію між фронтендом, бекендом, базою даних та компонентами машинного навчання із можливим масштабуванням чи зміною алгоритмів у кожній частині програми без втрати функціоналу.

Клієнтська частина додатку реалізована мовою JavaScript. Вибір саме цієї мови є закономірним, адже на сьогоднішній день вона є стандартом для створення браузерних розширень і інтерактивних інтерфейсів. JavaScript дозволяє відслідковувати взаємодію користувача з вебсторінкою в режимі реального часу: фіксувати збільшення шрифту, зміну контрастності, перемикання кольорової теми, масштабування сторінки та інші поведінкові параметри. Ці дані надсилаються на сервер, де вони можуть бути оброблені та використані для подальшого навчання моделі. Окрім цього, JavaScript забезпечує можливість миттєво застосовувати зміни інтерфейсу, які генерує користувач чи прогнозує модель: змінювати стилі, CSS-властивості або навіть перебудовувати окремі елементи DOM.

Серверна частина створена мовою Python. Python був обраний завдяки зручності інтеграції з бібліотеками машинного навчання, простоті розробки REST-API та високій швидкості прототипування. Ця мова має розвинену

екосистему для розробки і застосування машинного навчання та нейронних мереж. Екосистема включає безліч бібліотек та фреймворків як для обробки даних, наприклад Pandas та NumPy, так і для тренування моделей штучного інтелекту такі як Skitlearn, Keras та TensorFlow. Сервер відповідає за отримання та обробку поведінкових даних, взаємодію з базою даних, підготовку даних до навчання та виконання прогнозних запитів. Вся важка робота відбувається тут.

Такий підхід до побудови бекенду є типовим для інструментів аналізу даних і підтверджений практикою використання Python у роботах з NLP, розпізнавання зображень та аналізу транзакцій.

Для навчання та використання нейронної мережі було застосовано фреймворк TensorFlow. Цей фреймворк розроблений компанією Google, забезпечує можливість створення та навчання різних типів нейронних мереж зокрема і рекурентних нейронних мереж (зокрема LSTM або GRU), які працюють із послідовностями дій користувача і яку ми використовуємо у під час розробки браузерного розширення. TensorFlow дозволяє обробляти великі масиви даних та виконувати навчання на графічних прискорювачах, що є важливим для архітектур, які мають оновлювати прогностичну модель у міру накопичення нових поведінкових патернів. Бібліотека надає інструменти для перетворення навченої моделі у формат, оптимізований для швидкого виконання на сервері, що дозволяє отримувати прогноз за частки секунди.

Це особливо важливо де програма має миттєво, а можливо і у реальному часі реагувати на зміну поведінки користувача. Крім того, фреймворк підтримує подальше донавчання моделі на нових даних без потреби повного перенавчання моделі, що дозволяє додатку еволюціонувати разом з реальними сценаріями використання та підвищувати точність прогнозів у довгостроковій перспективі.

Для етапу кластеризації було застосовано K-Means, реалізований у складі бібліотеки Scikit-learn. Кластеризація дозволяє виділити групи користувачів зі схожими формами поведінки, наприклад, користувачів, які часто збільшують розмір шрифту, або тих, хто працює виключно у висококонтрастному режимі. Таким чином модель отримує додаткову інформацію про тип користувача, що

допомагає покращити точність прогнозування. Scikit-learn також використовується для реалізації дерев рішень, які допомагають формувати остаточні правила адаптації інтерфейсу, комбінуючи прогноз RNN із поточним станом користувача та кластерною приналежністю. Scikit-learn також дає можливість використовувати її як інструмент для побудови класичних моделей машинного навчання, наприклад у задачах фільтрації, класифікації чи виявлення аномалій.

Усі дані взаємодії зберігаються в SQL-базі даних. Вибір реляційної моделі зумовлений тим, що взаємодія користувача з інтерфейсом має структуру, яку легко формалізувати у вигляді таблиць: подія, час, параметри змін, ідентифікатор сесії та користувача. SQL забезпечує швидкий доступ до даних, можливість виконання складних вибірок і надійність зберігання. Для комплексу, яка постійно накопичує великі обсяги поведінкових записів, структурованість і можливість оптимізації запитів є критично важливими. Бази даних SQL також активно використовуються наприклад, у системах моніторингу, аналізу логів або генерації тест-планів, де зберігалася службова та структурована інформація. Але на сьогоднішній день розвиток нереляційних баз даних досяг такого рівня що за необхідності їх можна використати і подібного роду додатках.

Таким чином, вибраний набір програмних інструментів утворює гнучку та масштабовану архітектуру. JavaScript забезпечує інтерактивність та можливість збору поведінкових даних у браузері, Python та TensorFlow відповідають за обробку та прогнозування, Scikit-learn — за початковий аналіз і побудову логіки адаптації, тоді як SQL надійно зберігає інформацію, необхідну для навчання та роботи інфраструктури. Сукупність цих компонентів дозволяє створити ефективний додаток, здатного автоматично адаптувати інтерфейс під потреби користувача, мінімізуючи необхідність ручного налаштування та підвищуючи доступність вебресурсів для людей із порушеннями зору.

3.2 Комплексні вимоги доступності для користувачів із порушенням зору

Забезпечення доступності вебінтерфейсу для людей із порушенням зору є критичним компонентом розробки програмних рішень для людей із порушеннями. У більшості випадків вебресурси створюються без урахування потреб користувачів, які мають знижений зір, чутливість до світла, проблеми з фокусуванням або інші зорові порушення. Для того щоб розуміти, які саме адаптації браузерне розширення повинне робити, ми повинні визначити вимоги до функціонування системи. Щоб вважатись корисним для людини із порушенням зору інтерфейс повинен відповідати не лише рекомендаціям міжнародним настановам з доступності веб-контенту (WCAG) 2.2, а й міжнародним стандартам ISO, зокрема ISO 9241-171 “Guidance on Software Accessibility” та ISO/IEC 30071-1 “Accessible ICT Development Guidelines”. Нижче описано узагальнений комплекс вимог, згрупований за рівнями A, AA та AAA — аналогічно до WCAG, але доповнений положеннями ISO.

Вимоги рівня A

Базовий чи можна сказати мінімальний рівень, зосереджений на забезпеченні необхідної доступності, що дозволяє користувачеві з порушенням зору принаймні сприймати та використовувати вебінтерфейс.

По-перше, інформація не повинна передаватися лише кольором. Наприклад, повідомлення «поле заповнено невірно» не може бути позначене лише червоною межами, адже користувач із дальтонізмом або зниженим контрастним сприйняттям просто не відрізнить колір. У таких випадках розширення повинно автоматично додавати додаткові вказівники — іконки, текстові описи або збільшувати товщину рамки.

По-друге, усі зображення повинні мати альтернативні підписи які видими і зрозумілі. Це стосується банерів, кнопок із піктограмами тощо. Наприклад, якщо сайт використовує кнопку із зображенням лупи без тексту, розширення має

забезпечити можливість озвучення або показу мітки ARIA для користувача, який працює зі скрінрідером.

Важлива вимога є також можливість зупиняти будь-які рухомі, миготливі чи автоматично оновлювані елементи. WCAG обмежує тривалість такої активності до 5 секунд, після чого користувач повинен мати змогу зупинити прокручування елементу чи прибрати його. У реальному житті це стосується банерів, що прокручуються, рухомих рекламних блоків або автоматичних відеоплеєрів.

Рекомендації WCAG не містять вимоги про збільшенння клікабельних зони, але така вимога міститься в стандартах ISO. Згідно зі стандартом, кнопки навігації та керування, інтерактивні елементи повинні мати достатній розмір, щоб користувач із низькою точністю погляду, тремором або периферійним зором міг натиснути їх без помилок. Браузерне розширення має забезпечити можливість збільшення таких зон.

Крім цього, ISO рекомендує уникати перенавантаження інтерфейсу, тобто дрібного тексту, занадто багато декоративних елементів, непотрібне використання анімації, ці елементи можуть відволікати увагу споживача. Для прикладу, на сторінках новинних сайтів часто використовується велика кількість банерів, підкладок і другорядних елементів — розширення повинно мати режим «спрощеного перегляду», який зменшує кількість візуального шуму. Для людей із певними зоровими обмеженнями це може бути досить критично.

Вимоги рівня AA

Рівень AA передбачає підвищення можливостей доступності, на цьому рівні однією з ключових вимог є достатній контраст тексту та елементів інтерфейсу. WCAG вимагає співвідношення контрасту 4.5:1 для звичайного тексту та 3:1 для великого. Наприклад, світло-сірий текст на білому фоні, популярний у сучасному дизайні, не відповідає стандартам і може бути автоматично скоригований розширенням.

Другою важливою вимогою є можливість збільшення тексту до 200% чи навіть до 400% не має викликати порушення структури сторінки. Це означає, що при масштабуванні не повинно виникати накладання блоків, зсуву кнопок або втрати контенту, простіше кажучи верстка сторінки не має сильно ламатись. Розширення має коригувати міжрядкові інтервали, переносити блоки та адаптувати структуру вигляду сторінки.

Окремо слід підкреслити вимогу Reflow — зручність перегляду вмісту при звуженні ширини області перегляду до 320 px (дисплей мобільного екрана). Якщо користувач занадто збільшить масштаб сторінки це буде критично: інтерфейс повинен автоматично переходити в одноколонковий режим без горизонтального прокручування.

Стандарти ISO додають необхідність підтримки перемикання колірних схем — світлої, темної, висококонтрастної — і навіть інверсії кольорів, що дуже корисно для людей зі світло чутливістю або дегенеративними змінами сітківки. Наприклад, чорний текст на білому фоні для таких користувачів може бути болісним; розширення повинно дозволяти змінити схему одним натисканням.

Також в стандарті ISO наголошується на підтримці користувацьких шрифтів та можливості змінювати міжрядкові інтервали, відстані між словами та літерами. У практиці це допомагає людям із астигматизмом або труднощами фокусування: текст із більшою відстанню між рядками та відстанню між буквами читається для таких людей значно легше.

Вимоги рівня AAA

І так найвищий рівень доступності — AAA який орієнтований на користувачів із складними випадками порушеннями зору, для яких стандартні налаштування є недостатніми і стандартний підхід може і не задовольняти повністю їх потреби. Тому цей рівень і пояснює важливість персоналізації підходу до адаптації інтерфейсу.

WCAG на даному рівні вимагає наявність можливості налаштування підвищеного контрасту 7:1, що дозволить користувачам із дуже низьким зором

читати текст без додаткових засобів. У браузерному розширенні це може бути реалізовано окремим режимом, наприклад «Режим ультра контрасту» або забезпечити точне виставлення комфортного рівня контрасту.

На даному рівні стандарти вимагають повної заборони використання зображень тексту тобто, якщо сайт містить текстові банери або кнопки з написами у вигляді PNG, розширення повинно перетворювати такі елементи у реальний текст (через OCR або створення альтернативного опису).

Вимоги ISO особливо важливі на цьому рівні. Насамперед стандарт вимагає адаптації враховуючи контекст середовища в якому знаходиться споживач. Додаток повинен розуміти, у якому контексті перебуває користувач — читає статтю, заповнює форму, переглядає відео — і автоматично пропонувати оптимальні параметри відображення. Наприклад, при заповненні форми розширення може збільшити поля вводу та підсилити їх рамки.

Окрім цього, ISO рекомендує використовувати персональні профілі доступності, що на практиці означає, що система повинна запам’ятовувати всі налаштування користувача й застосовувати їх на будь-якому сайті автоматично.

Найбільш інноваційною вимогою ISO є можливість автоматичного навчання системи адаптації. У проєкті це реалізовано через рекурентні нейронні мережі (RNN), які навчаються аналізувати дії користувача (збільшення тексту, зміна кольорів, відключення анімацій) та прогнозувати, які саме параметри необхідно змінювати. Наприклад, якщо користувач завжди збільшує текст на новинних сайтах, модель з часом почне робити це автоматично.

Остання вимога яка стосується зниження когнітивного навантаження. Інтерфейс повинен бути максимально простим, відсутні зайві елементи, має бути присутня чітка видимість фокуса та велика ієрархія заголовків. Для користувача із суттєвими проблемами зору складні сторінки можуть бути фізично та ментально складними для сприйняття, тому розширення повинно пропонувати спрощений режим перегляду.

3.3 Реалізація браузерного розширення для автоматизованої адаптації інтерфейсу для людей із порушенням зору

Архітектура рішення має модульну структуру, що складається з трьох основних компонентів, а саме клієнтського браузерного розширення, серверної частини, модуля машинного навчання.

Браузерне розширення відповідає за збір подій, фіксацію дій користувача, попередню обробку даних, застосування адаптацій і взаємодію з DOM-структурою вебсторінки. Розширення працює у фоновому сервісі, перехоплює взаємодії з інтерфейсом і через REST API надсилає пакети даних на сервер.

Фіксація користувацької активності здійснюється для покращення якості сервісу, персоналізації взаємодії та діагностики роботи програмного забезпечення. Опишемо дані які фіксуються і як ці дані нам допоможуть у досягнення цілі автоматизованої адаптації інтерфейсу.

Перш за все фіксується зміна розміру тексту, та розміру вікна. Потрібно зрозуміти як адаптувати контент для під конкретні потреби. Потрібно розуміти коли користувач змінює масштаб сторінки, варто врахувати контекст, адже при різних умова цей параметр може бути різний. Розмір шрифту важливий і дозволяє зрозуміти де в майбутньому потрібно його збільшити і зменшити. Чому не можна застосувати фіксований розмір сторінки чи шрифту на всіх веб-сторінках, адже це швидше і простіше? Кожен вебресурс має свій стиль і тип подання інформації. Сайти з новинами це зазвичай велики обсяги тексту на екрані. Вони орієнтовані на швидке сканування статей, інтернет-магазини навпаки більше зосереджені на візуальних елементах. Тому використання універсальних розмірів шрифтів призведе до порушення верстки і втрати частини функціоналу.

Часте перемикання між вкладками і часте повернення до попередніх розділів може вказувати на втому очей чи складність концентрації уваги, у такому випадку допоможе збільшення міжрядкового інтервалу або шрифту.

Для налаштування доступності фіксується корекція контрасту, увімкнення спрощеного режиму, керування анімацією.

Також розширення фіксує контекст взаємодії. Система спостерігає чи працює користувач із текстовою статтею, відео, інтернет-магазином чи соціальною мережею. Для кожного типу контенту можуть знадобитись різні типи налаштувань. Важливими даними є час перебування та активність під час доби. Проведений час допоможе спрогнозувати оптимальний час застосування адаптацій, активність протягом дня важлива при застосуванні деяких адаптацій які користувач вмикає в залежності від часу доби.

Для забезпечення доступності згідно стандартів було реалізовано інтерфейс представлений на рисунку 3.1, де налаштування для кожного рівня винесене в окрему вкладку. Кожна вкладка відповідає за налаштування певного рівня.

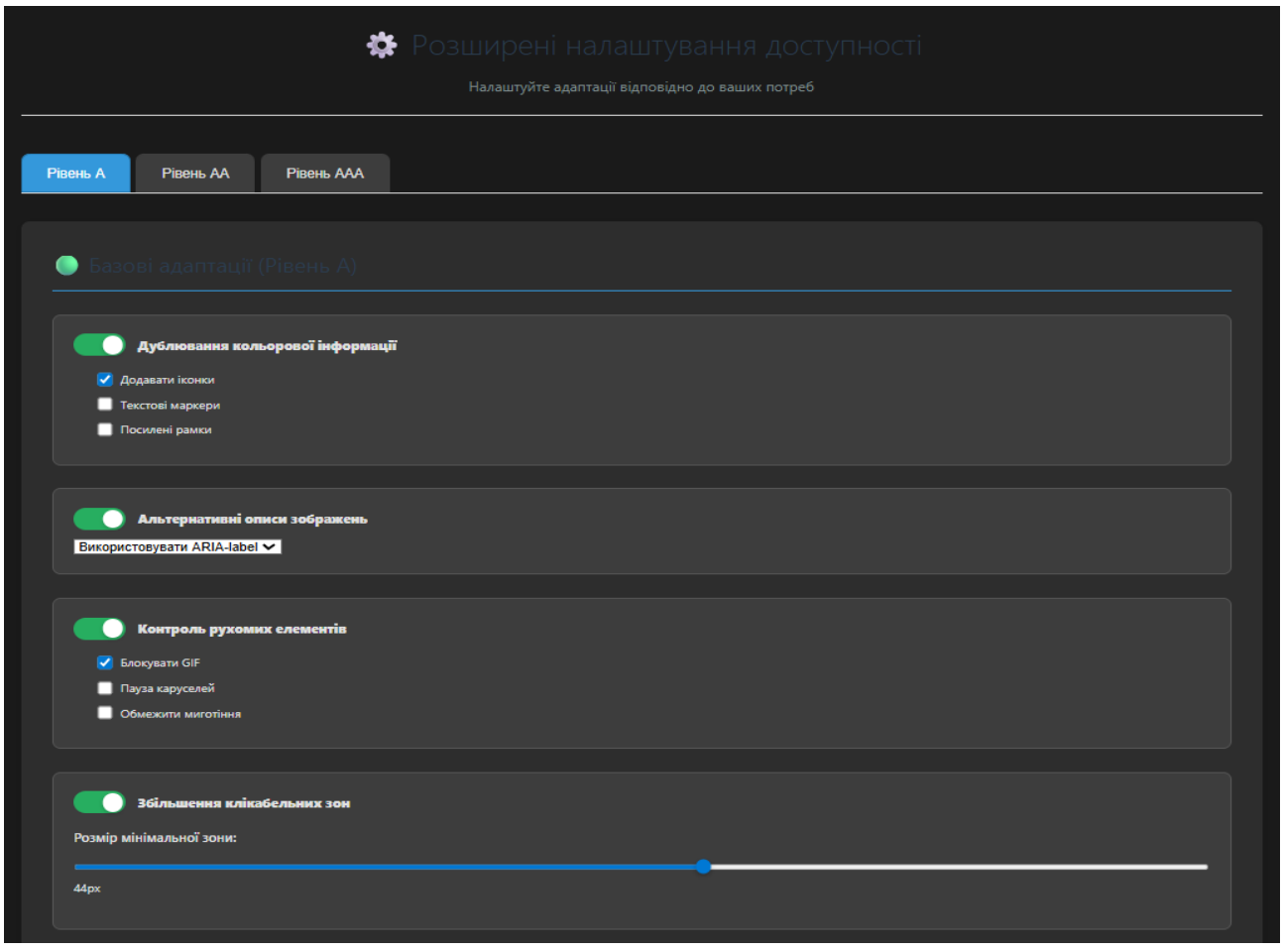


Рис. 3.1 Розширені налаштування браузерного розширення адаптації

Для передачі зібраних даних на сервер використовується об’єкт у форматі json. Приклад такого об’єкту представлений на рисунку 3.2.

```
{"sessionId": "session_1705325425123_abc123def",
 "userId": "user_123456789",
 "timestamp": "2024-01-15T14:30:25.123Z",
 "currentSettings": {
   "colorDuplication": true,
   "altText": true,
   "animationControl": true,
   "largeTargets": true,
   "simplifiedView": false,
   "autoContrast": true,
   "textZoom": true,
   "reflowMode": true,
   "typographySettings": false,
   "ultraContrast": false,
   "textOcr": false,
   "contextAdaptation": true,
   "cognitiveSimplification": false,
   "targetSize": 44,
   "contrastLevel": 2,
   "fontSize": 125,
   "lineHeight": 1.5,
   "letterSpacing": 0,
   "wordSpacing": 0,
   "fontFamily": "system",
   "currentTheme": "light"
 },
 "settingsChanges": [
   {"timestamp": "2024-01-15T14:05:00.000Z",
     "changedSettings": {
       "fontSize": 125,
       "contrastLevel": 2},
     "reason": "manual_adjustment"}
 ],
 "context": {
   "pageType": "news",
   "userContext": "reading"
 }}
```


Рис. 3.2 Об’єкт із налаштуваннями юзера зібраний браузерним розширенням

Наступна частина браузерного розширення це сервер який приймає, зберігає і оброблює дані та далі передає в модуль із машинним навчанням. Об’єкт *json* який представлений на рисунку 3.2 із зібраними даними поведінки юзера приходить по REST API і зберігається у базі даних. Дані про поведінку користувача та налаштування які користувач встановлює самостійно оброблюються різними класами. Перед обробкою сервер перевіряє: чи є у повідомленні ідентифікатор сесії, масив подій та час відправки. Якщо все гаразд — дані проходять далі. Кожна взаємодія з інтерфейсом прив’язана до конкретної користувацької сесії.

Якщо сервер бачить, що сесія з таким sessionId ще не існує, він створює новий запис — зберігає IP-адресу, User-Agent браузера та (за потреби) ідентифікатор користувача. Це важливо, тому що надалі модель аналізує поведінку всередині конкретних сесій та за окремими користувачами. Далі сервер проходить усі записи в полі data. Кожен елемент списку — це одна дія або один "знімок стану" інтерфейсу.

Для кожного такого запису створюється окремий об’єкт у таблиці BehaviorData, куди заносяться: час події, тип взаємодії, налаштування інтерфейсу (шрифт, контраст, тема, масштаб, міжрядковість тощо), поведінкові метрики (глибина скролу, швидкість, кількість кліків), контекст (тип сторінки, елемент у фокусі, URL), параметри екрана. За зберігання даних поведінки відповідає клас *BehaviorData* за збереження налаштувань користувача клас *UserSettings*.

Приклад збережених даних поведінки користувача у базі даних представлений на рисунку 3.3, а дані налаштувань користувача – на рисунку 3.4.

Усе це зберігається у SQLite-базі. Це — “сирі” дані, на основі яких модель згодом навчається прогнозувати майбутні потреби користувача.

Окремий потік у застосунку регулярно (наприклад, раз на годину) викликає функцію оновлення моделі. Функція фонового оновлення представлена на рисунку 3.5.

Першим кроком вона вибирає всі поведінкові записи за останні 7 днів. Якщо назбиралося менше ніж 100 подій — сенсу навчати модель немає, тому процес відкладається.

Коли даних достатньо, вони перетворюються у таблицю *pandas.DataFrame*. Це зручний формат для подальшої обробки — інженерії ознак, нормалізації, побудови послідовностей тощо.

session_id	user_id	timestamp	color_duplication	alt_text	animation_control	large_targets	simplified_view	auto_contrast
S001	U001	2025-01-01 10:12:45	0	1	1	1	1	1
S001	U001	2025-01-01 10:14:32	0	1	1	1	1	1
S001	U001	2025-01-01 10:20:02	0	1	1	1	1	1
S002	U001	2025-01-02 09:05:12	0	1	1	0	0	1
S002	U001	2025-01-02 09:07:33	0	1	1	1	1	1
S003	U002	2025-01-05 18:32:11	0	1	1	1	1	1
S003	U002	2025-01-05 18:40:50	0	1	1	1	1	1
S004	U003	2025-01-07 21:14:03	0	1	1	1	0	0
S004	U003	2025-01-07 21:18:59	0	1	1	1	1	1
S005	U004	2025-01-10 12:55:47	0	1	1	1	1	1

Рис. 3.3 Приклад даних поведінки користувача у базі даних.

session_id	timestamp	event_type	page_type	user_context	font_size	contrast_level	zoom_level	theme	line_height
S1001	2025-01-12 10:04:12	font_change	news	reading	1.6	70	1.2	dark	1.8
S1001	2025-01-12 10:05:22	contrast_change	news	reading	1.6	85	1.2	dark	1.8
S1001	2025-01-12 10:06:35	scroll	news	reading	1.6	85	1.2	dark	1.8
S1204	2025-01-12 12:14:09	zoom_change	ecommerce	shopping	1.4	60	1.4	light	1.6
S1204	2025-01-12 12:14:55	click	ecommerce	shopping	1.4	60	1.4	light	1.6
S1204	2025-01-12 12:16:00	scroll	ecommerce	shopping	1.4	60	1.4	light	1.6
S2002	2025-01-12 15:22:41	font_change	banking	form_filling	1.8	90	1.5	dark	2.0
S2002	2025-01-12 15:23:10	keypress	banking	form_filling	1.8	90	1.5	dark	2.0
S2002	2025-01-12 15:24:05	scroll	banking	form_filling	1.8	90	1.5	dark	2.0
S3007	2025-01-12 18:44:22	theme_change	article	reading	2.0	95	1.3	dark	2.2

Рис. 3.4 Приклад даних збережених налаштувань користувача у базі даних.

```python
def update_model(app):
    """Оновлення моделі на нових даних"""
    try:
        (variable) week_ago: datetime
        кдень
        week_ago = datetime.utcnow() - timedelta(days=7)

        behavior_data = BehaviorData.query.filter(
            BehaviorData.timestamp >= week_ago
        ).all()

        if len(behavior_data) < 100:  # Мінімальна кількість даних
            print("Недостатньо даних для оновлення моделі")
            return

        # Конвертуємо в DataFrame
        data_dicts = [data.to_dict() for data in behavior_data]
        df = pd.DataFrame(data_dicts)

        # Оновлюємо модель
        app.model_manager.train_complete_pipeline(df)
        print("Модель успішно оновлено")
```

Рис. 3.5 Функція збіру даних для тренування

На рисунку 3.5 можна побачити що дані поведінки користувача збираються за тиждень, і мінімальна кількість таких даних має бути більше 100, бо інакше тренування моделі буде менш ефективним. Чим більше даних тим точніше буде натренована модель. Для відправки для подальшої обробки дані із бази даних ми конвертуємо усе в таблицю датафрейм користуючись можливостями бібліотеки pandas. Сконвертована таблиця далі відправляється у функцію *train_complete_pipeline* де буде відбуватись основна робота по нормалізації і тренуванню моделі.

Потрапивши до функції *train_complete_pipeline* до даних застосовується модуль *FeatureEngineer* - це процес перетворення «сирих» поведінкових даних користувача на інформативні змінні, які дозволяють моделі виявляти закономірності. У системі автоматизованої адаптації інтерфейсу це необхідно,

оскільки події, що надходять від браузерного розширення, містять лише базові значення (наприклад, розмір шрифту, контрастність чи глибину скролу), які самі по собі малоінформативні.

Наприклад:

```
fontSize = 120
scrollDepth = 80
pageType = "news"
```

Дані які наведені вище не дають моделі розуміння контексту та тенденцій. Тому дані доповнюються додатковими часовими даними, статистичними ознаками, контекстними ознаками. Ці ознаки роблять дані уже більш структурованими і зрозумілими. Вони допомагають моделі отримувати не окремі цифри, але і поведінкові патерни, що значно підвищує точність прогнозування. Наприклад, ознака *font_size_change = df['font_size'].diff()* показує, наскільки часто користувач збільшує текст, а *reading_efficiency=scroll_depth / (time_on_page + 1)* демонструє комфортність читання.

Після виконання функцій *add_temporal_features(self, df), add_statistical_features(self, df)* тощо, дані отримують додатковий контекст і набувають вигляду наприклад:

```
hour_of_day = 12
scroll_depth_mean_5 = 94.5
page_type_news = 1
```

На цьому етапі дані перестають бути просто набором сирих полів і перетворюються на структуру, з якої модель може витягувати закономірності.

Наступний крок нормалізація — етап попередньої обробки, під час якого всі числові ознаки приводяться до одного масштабу. Функція нормалізації представлена на рисунку 3.6

```python
def fit(self, df):
    """Навчання нормалізаторів на тренувальних даних"""
    # Визначаємо типи features
    (variable) categorical_features: list _features(df)
    categorical_features = self._get_categorical_features(df)
    boolean_features = self._get_boolean_features(df)

    # Нормалізація числових features
    for feature in numerical_features:
        scaler = StandardScaler()
        df[feature] = scaler.fit_transform(df[[feature]])
        self.scalers[feature] = scaler

    # Кодування категоріальних features
    for feature in categorical_features:
        encoder = LabelEncoder()
        df[feature] = encoder.fit_transform(df[feature].astype(str))
        self.encoders[feature] = encoder

    # Обробка булевих features
    for feature in boolean_features:
        df[feature] = df[feature].astype(int)

    self.feature_names = list(df.columns)
    return df
```

Рис. 3.6 Функція нормалізації класу *DataNormalizer*

Це необхідно, тому що дані які приходять від клієнта дуже різнорідні, наприклад *scrollDepth* може бути в діапазоні від 0-5000, а *font_size* може змінитись від 80-200. Без нормалізації велике число (наприклад, *scroll_depth = 3000*) буде вважатись більшим над невеликим (*contrast_level = 3*) лише тому, що воно більше за масштабом, а не тому, що важливіше. Завдання класу *DataNormalizer* — привести всі ознаки до єдиного числового вигляду і масштабу, роблячи їх співставними. Він автоматично визначає тип кожної ознаки: числову, категоріальну або булеву.

Числові ознаки стандартизуються за допомогою *StandardScaler*, який перетворює значення так, що вони мають середнє 0 та стандартне відхилення 1.

Категоріальні значення, наприклад *page_type: 'news'*, кодуються у цілі числа через *LabelEncoder*, оскільки нейронні мережі не можуть інтерпретувати текст. Булеві значення (*simplified_view, animation_state*) також переводяться у 0 або 1. Приклад даних до і після нормалізації наведені в таблиці 2.1.

Таблиця 2.1

Порівняльна таблиця даних до і після нормалізації

Дані до нормалізації	**Дані після нормалізації**
font_size = 1.6	*font_size = -0.61*
contrast_level = 70	*contrast_level = -0.44*
scroll_depth = 0.12	*scroll_depth = -0.81*
scroll_speed = 0.05	*scroll_speed = -0.66*
click_count = 2	*click_count = 2*
time_on_page = 45.3	*time_on_page = 45.3*
page_type = "news"	*page_type = "news"*
theme = "dark"	*theme = "dark"*
simplified_view = 1	*simplified_view = 1*

Тобто один «логічний» запис перетворюється на набір нормалізованих чисел. Числові ознаки тепер мають середнє близьке до нуля та зіставні масштаби, а текстові значення *page_type* і *theme* замінені на цілі числа за правилами *LabelEncoder*.

Після того як дані проходять етап нормалізації, вони набувають уніфікованого числового вигляду, у якому всі ознаки мають однакову шкалу та можуть бути коректно інтерпретовані моделлю машинного навчання. Приклад даних які приходять після нормалізації для подальшого формування послідовностей представлений на рисунку 3.7.

```
=== Нормалізовані дані для навчання ===
          id  session_id  user_id            timestamp  ...  font_size_by_page_type  contrast_by_context  reading_efficiency  interaction_intensity
0  -1.723412           0        0  2025-01-21 09:10:10  ...                1.651419             0.621341           -0.845607              -0.448949
1  -1.706091           0        0  2025-01-21 09:12:30  ...                2.239927             0.824539           -0.470494              -0.448949
2  -1.688771           0        0  2025-01-21 09:15:05  ...                2.239927             1.230936            0.140383              -0.448949
3  -1.671450           1        1  2025-01-21 10:30:05  ...               -0.310274            -1.309046           -0.773634               2.150380
4  -1.654129           1        1  2025-01-21 10:32:20  ...               -0.310274            -1.309046            0.163642               1.822226
5  -1.636808           1        1  2025-01-21 10:34:10  ...               -0.310274            -1.309046            0.004555               3.719487
6  -1.619488           2        2  2025-01-21 11:05:40  ...                0.212844             1.027738           -1.496623              -0.448949
7  -1.602167           2        2  2025-01-21 11:08:15  ...                0.212844             1.027738           -0.459868              -0.448949
8  -1.584846           2        2  2025-01-21 11:12:00  ...                0.212844             1.027738            0.103507              -0.448949
9  -1.567526           3        3  2025-01-21 11:55:30  ...               -0.604528            -0.801049           -1.430756              -0.448949
10 -1.550205           3        3  2025-01-21 11:57:05  ...               -0.604528            -0.801049           -1.479263              -0.448949
11 -1.532884           3        3  2025-01-21 11:58:40  ...               -0.604528            -0.801049           -1.499649               2.082990
12 -1.515563           4        4  2025-01-21 12:50:20  ...               -0.931477            -0.191454           -0.261932              -0.448949
13 -1.498243           4        4  2025-01-21 12:51:10  ...               -0.931477            -0.191454           -0.390493               2.790973
14 -1.480922           4        4  2025-01-21 12:52:45  ...               -0.931477            -0.191454            2.219580              -0.448949
15 -1.463601           5        5  2025-01-22 09:11:10  ...                1.651419             0.621341           -0.802368              -0.448949
16 -1.446281           5        5  2025-01-22 09:13:10  ...                1.945673             0.824539           -0.225955              -0.448949
17 -1.428960           5        5  2025-01-22 09:14:55  ...                1.945673             1.230936            0.689972              -0.448949
18 -1.411639           6        6  2025-01-22 09:26:10  ...               -0.310274            -1.309046           -0.769835               2.160301
19 -1.394318           6        6  2025-01-22 09:27:50  ...               -0.310274            -1.309046            0.042424              -0.448949
```

Рис. 3.7 Нормалізовані дані для навчання

Наступний етап — формування структурованих послідовностей та підготовка таргетів, які використовуються для навчання рекурентної нейронної мережі (LSTM). Саме на цьому етапі дані перетворюються з окремих рядків таблиці поведінки у змістовні послідовності, що відображають поведінку користувача в часі. Цю роботу виконує функція *create_sequences* представлена на рисунку 3.8.

```python
def create_sequences(self, df, user_id):
    """Створення послідовностей для LSTM"""
    sequences = []
    targets = []

    # Групуємо за користувачем
    user_data = df[df['user_id'] == user_id].sort_values('timestamp')
    if len(user_data) < self.sequence_length:
        print(f"Недостатньо даних для користувача {user_id} для створення послідовностей.")
        return sequences, targets

    # Створюємо послідовності
    for i in range(len(user_data) - self.sequence_length):
        sequence = user_data.iloc[i:i + self.sequence_length]
        target = user_data.iloc[i + self.sequence_length]

        # Features для послідовності (поведінка + контекст)
        sequence_features = self._extract_sequence_features(sequence)

        # Target (налаштування)
        target_features = self._extract_target_features(target)

        sequences.append(sequence_features)
        targets.append(target_features)

    return np.array(sequences), np.array(targets)
```

Рис. 3.8 Функція створення послідовностей для навчання моделей

Спочатку функція відбирає з усієї таблиці лише ті рядки, що належать одному конкретному користувачу. Такий відбір потрібний, оскільки LSTM навчається на індивідуальних патернах взаємодії кожної людини. Після цього результуючі дані сортуються за часом (timestamp), щоб послідовність подій йшла у правильному хронологічному порядку. Це критично важливо, оскільки LSTM працює саме з часовими залежностями і повинна бачити, які дії виконувалися раніше, а які — пізніше.

Далі функція перевіряє, чи має користувач достатньо даних для формування послідовності. Наприклад, якщо для створення одного вікна LSTM потрібно три події, а в користувача є лише дві, то з таких даних послідовність побудувати неможливо. У такій ситуації функція повертає порожній результат.

Якщо даних достатньо, функція починає рухатися по них «ковзаючим вікном». На кожному кроці вона бере певну кількість послідовних записів — це і є один приклад поведінкової послідовності для LSTM. Наступний запис після цього вікна використовується як «цільове значення», тобто те, що модель повинна навчитися передбачати. Наприклад, послідовність може містити інформацію про прокрутку сторінки, зміну контрасту і збільшення шрифту, а ціллю може бути те, що користувач увімкнув спрощений режим перегляду.

Після формування чергового фрагмента даних функція витягує з нього ключові особливості — зміни в розмірі шрифту й інші параметри. Ці обчислені характеристики і стають готовими LSTM-векторами для навчання. Окремо з цільового рядка витягується інформація про те, які налаштування інтерфейсу були застосовані користувачем, — це і є значення, яке має передбачити модель.

Після виділення послідовностей система виконує агрегацію ознак усередині кожного вікна. Замість передачі необроблених подій у модель, обчислюються узагальнені характеристики: середня глибина прокручування, сумарна кількість кліків, середня швидкість скролу, останні значення параметрів інтерфейсу (розмір шрифту, контрастність), а також зміни між подіями.

Такий підхід дозволяє моделі бачити не лише одиничні дії, а тенденції, що формуються в процесі взаємодії.

Кожна сформована пара «послідовність → прогнозовані налаштування» додається до підсумкового набору тренувальних даних. Наприкінці функція перетворює всі послідовності і таргети на NumPy-масиви, які потім передаються в модель для навчання.

Кожна така послідовність відображає частину взаємодії користувача в певний проміжок часу. Наприклад, якщо користувач прочитав статтю, кілька разів збільшив шрифт, змінив контрастність, усі ці дії потраплять у вікно, з якого модель робитиме прогноз.

Паралельно з цим формується вектор цільових значень (таргет), який LSTM повинна навчитися передбачати. Таргет містить ті параметри інтерфейсу, які система має рекомендувати користувачу в майбутньому: оптимальний розмір шрифту, рівень контрастності, необхідність увімкнення спрощеного режиму, відключення анімацій чи збільшення інтерактивних областей. Таким чином, кожна послідовність поведінкових даних перетворюється на пару: вхідний вектор (X) та прогнозований результат (y). Приклад отриманих послідовностей і таргетів представлені на рисунку 3.9 та 3.10.

```
=== Підготовлені послідовності для навчання ===
sequences [array([[-0.84107195, -0.19305152,  0.       , -0.19737205,  0.02788171,
         0.08401721,  5.        ,  3.        ],
       [-0.23452968, -0.39414685,  0.        ,  0.60673632,  0.83604736,
         0.45413709,  5.        ,  3.        ],
       [ 0.7763741 ,  0.20913915,  0.        ,  1.28552909,  0.83604736,
         1.19437684,  5.        ,  3.        ]]), array([[-0.23452968, -0.39414685,  0.        ,  0.60673632,  0.83604736,
         0.45413709,  5.        ,  3.        ],
       [ 0.7763741 ,  0.20913915,  0.        ,  1.28552909,  0.83604736,
         1.19437684,  5.        ,  3.        ],
       [-0.84107195, -0.19305152,  0.        , -0.75084924,  0.02788171,
         0.08401721,  5.        ,  3.        ]]), array([[ 0.7763741 ,  0.20913915,  0.        ,  1.28552909,  0.83604736,
         1.19437684,  5.        ,  3.        ],
       [-0.84107195, -0.19305152,  0.        , -0.75084924,  0.02788171,
         0.08401721,  5.        ,  3.        ],
       [-0.31540198,  0.00804381,  0.        , -0.51066103,  0.02788171,
         0.23206516,  5.        ,  3.        ]]), array([[-0.84107195, -0.19305152,  0.        , -0.75084924,  0.02788171,
         0.08401721,  5.        ,  3.        ],
       [-0.31540198,  0.00804381,  0.        , -0.51066103,  0.02788171,
         0.23206516,  5.        ,  3.        ],
       [-0.31540198,  0.00804381,  0.        , -0.27047282,  0.83604736,
         0.82425696,  5.        ,  3.        ]]), array([[-0.31540198,  0.00804381,  0.        , -0.51066103,  0.02788171,
         0.23206516,  5.        ,  3.        ],
       [-0.31540198,  0.00804381,  0.        , -0.27047282,  0.83604736,
         0.82425696,  5.        ,  3.        ],
       [ 0.97855486,  0.20913915,  0.        , -0.04072757,  0.83604736,
         0.82425696,  5.        ,  3.        ]]), array([[-0.92194425,  0.61132981,  1.        , -0.74040628, -0.78028393,
        -1.02634241,  5.        ,  3.        ],
       [-0.03234892,  1.41571114,  1.        , -0.34357358, -0.78028393,
        -1.02634241,  5.        ,  3.        ],
       [-0.03234892,  1.41571114,  2.        , -0.0616135 , -0.78028393,
        -1.02634241,  5.        ,  3.        ]]), array([[-0.03234892,  1.41571114,  1.        , -0.34357358, -0.78028393,
        -1.02634241,  5.        ,  3.        ],
```

Рис. 3.9 Приклад отриманих послідовностей (sequences)

```
targets [[0.43196454 0.67620901 1.         1.         ]
 [0.83604736 0.97230491 1.         1.         ]
 [0.83604736 0.97230491 1.         1.         ]
 [0.83604736 0.97230491 1.         1.         ]
 [0.83604736 1.41644876 1.         1.         ]]
```

Рис. 3.10 Приклад отриманих таргетів (targets).

Заключним етапом є формування підсумкових навчальних масивів (векторів ознак). Після генерації всіх послідовностей дані збираються у дві матриці: *X* — набір вхідних ознак, та *y* — набір таргетів. Модель отримує їх у вигляді багатовимірних тензорів, у яких перший вимір — це кількість послідовностей, другий — довжина часової історії, а третій — набір ознак усередині кожної події. Після цього дані розділяються на навчальну та валідаційну вибірки, що дозволяє ефективно контролювати якість навчання. Приклади сформованих векторі ознак поведінки користувача (*X*) та майбутніх налаштувань (*y*), наведені на рисунку 3.11 та 3.12.

```
[
 [
  [
   -0.841071945609643,
   -0.1930515195806641,
   0.0,
   -0.1973720542904503,
   0.027881714744801778,
   0.08401721170456498,
   5.0,
   3.0
  ],
  [
   -0.2345296771411505,
   -0.39414685247718945,
   0.0,
   0.6067363150410139,
   0.8360473595216237,
   0.4541370870550699,
   5.0,
   3.0
  ],
```

Рис. 3.11 Приклад вектору ознак поведінки користувача

```
y: [[ 0.02788171  0.08401721  1.          1.        ]
 [ 0.02788171  0.23206516  1.          1.        ]
 [ 0.83604736  0.82425696  1.          1.        ]
 [ 0.83604736  0.82425696  1.          1.        ]
 [ 0.83604736  1.19437684  1.          1.        ]
 [-0.37620111 -0.65622254  0.          0.        ]
 [-0.37620111 -0.50817459  0.          0.        ]
 [-0.37620111 -0.50817459  0.          0.        ]
 [-0.37620111 -0.50817459  0.          0.        ]
 [ 0.43196454  0.45413709  0.          0.        ]
 [-0.78028393 -1.02634241  1.          0.        ]
 [-0.78028393 -1.02634241  1.          0.        ]
 [-0.78028393 -1.02634241  1.          0.        ]
 [-0.78028393 -1.02634241  1.          0.        ]
 [-0.78028393 -1.02634241  1.          0.        ]]
```

Рис. 3.12 Приклад вектору ознак майбутніх налаштувань

Модель машинного навчання не може навчатися і тестуватися на одних і тих самих прикладах, тому перед початком тренування набір даних потрібно поділити на *train* і *validation*. Саме це і робить частина коду представлена на рисунку 3.12.

```python
# Розділення на train/validation
split_idx = int(0.8 * len(X))
X_train, X_val = X[:split_idx], X[split_idx:]
y_train, y_val = y[:split_idx], y[split_idx:]
```

Рис. 3.13 Процес ділення даних на тренувальні та валідаційні

Спочатку обчислюється індекс *split_idx*, який визначає точку, у якій потрібно «розрізати» масив даних. Оскільки у проєкті використовується співвідношення 80/20, 80% перших послідовностей підуть на навчання моделі, а решта 20% — на її валідацію. Після визначення цього індексу відбувається сам поділ. Перша частина *X_train* береться від початку масиву X до позиції split_idx, а друга частина *X_val* — від *split_idx* і до кінця. Точно так само ділиться й набір правильних відповідей у перші значення утворюють *y_train*, а решта — *y_val*.

Важливо, що *X* і y діляться абсолютно синхронно, щоб кожна послідовність у *X* завжди відповідала «правильній відповіді» в *y*.

Після виконання цього коду з'являється два набори даних. Перший (*X_train, y_train*) використовується для того, щоб модель навчилася знаходити закономірності. Другий (*X_val, y_val*) дозволяє перевірити, чи може модель робити правильні прогнози на нових даних, які вона не бачила під час навчання. Це дозволяє оцінити реальну точність моделі і запобігти перенавчанню.

Наступний етап - побудова та тренування моделі. Почнемо із побудови. Спочатку викликається метод *build_model()*, якому передаються два параметри: кількість вхідних ознак і кількість вихідних значень. У виклику використовується *X_train.shape[2]* та *y_train.shape[1]*. Код функції побудови моделі представлений на рисунку 3.13.

Опишемо етапи побудови і параметри моделі детальніше.

Функція *build_model(self, input_dim, output_dim)* будує й налаштовує архітектуру LSTM-моделі. Змінна *input_dim* — скільки ознак має один крок послідовності (у тебе це 8 фіч на крок). Змінна *output_dim* — скільки значень ми хочемо передбачити (наприклад 4–5 параметрів інтерфейсу). Модель поділена на шари які йдуть один за одним.

Перший LSTM має 128 нейронів і параметр *return_sequences=True*, який означає, що він повертає послідовність прихованих станів для кожного кроку, а не тільки останній. Тобто LSTM зчитує всю історію дій користувача і для кожного кроку формує вектор із 128 значень, у якому зашифрована інформація і про цей крок, і про попередній контекст. Регуляризатори *kernel_regularizer=L2(0.001)* і *recurrent_regularizer=L2(0.001)* додають L2-штраф до ваг як для звичайних зв'язків, так і для рекурентних. Це робиться, щоб обмежувати занадто великі ваги та зменшувати перенавчання — особливо важливо при невеликій кількості даних.

Після LSTM застосовується *BatchNormalization()*. Він «вирівнює» розподіл активацій усередині пакета, стабілізує градієнти, прискорює навчання й робить його більш стійким.

Далі йде *Dropout(0.3)*: на кожному кроці навчання випадково «вимикається» 30% нейронів шару. Це теж захист від перенавчання — мережа не може «запам’ятати» дані одними й тими ж шляхами й змушена шукати більш загальні закономірності.

```
def build_model(self, input_dim, output_dim):
    """Побудова архітектури LSTM моделі"""
    self.input_dim = input_dim
    self.output_dim = output_dim

    self.model = Sequential([
        # Перший LSTM шар
        LSTM(128,
             return_sequences=True,
             input_shape=(self.sequence_length, input_dim),
             kernel_regularizer=l2(0.001),
             recurrent_regularizer=l2(0.001)),
        BatchNormalization(),
        Dropout(0.3),

        # Другий LSTM шар
        LSTM(64,
             return_sequences=False,
             kernel_regularizer=l2(0.001),
             recurrent_regularizer=l2(0.001)),
        BatchNormalization(),
        Dropout(0.3),

        # Повнозв'язні шари
        Dense(32, activation='relu', kernel_regularizer=l2(0.001)),
        BatchNormalization(),
        Dropout(0.2),

        Dense(16, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.1),

        # Вихідний шар
        Dense(output_dim, activation='linear')
    ])

    # Компіляція моделі
    self.model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='mse',
        metrics=['mae', 'mse']
    )

    return self.model
```

Рис. 3.14 Код функції побудови моделі LSTM.

Другий LSTM отримує вже оброблену послідовність із першого шару (кожен крок — вектор розміром 128). Тут *return_sequences=False*, отже на виході ми беремо лише останній прихований стан. Це вектор на 64 елементи, який можна вважати підсумком усієї послідовності поведінки користувача. У ньому вже стиснута й узагальнена інформація про те, що відбувалося на всіх кроках.

І знову — нормалізація батчів та *Dropout* для стабільного й не перенавченого навчання.

Після другого LSTM маємо вектор із 64 чисел. Цей вектор — «високорівнева» ознака послідовності. Далі ти пропускаєш його через два *Dense*-шари, які поводяться як класична багатошарова нейронка. Тут використовується активація *ReLU* (Rectified Linear Unit). На це є декілька причин: *ReLU* добре працює з глибокими мережами й менше страждає на проблему зникнення градієнта, ніж *tanh* чи *sigmoid*. Модель може вчити складні залежності між поведінкою користувача й майбутніми налаштуваннями тобто дає нелінійність. Обчислюється функція дуже швидко: *max(0, x)* — проста операція, що важливо для ефективного тренування, а також природньо створює «розріджені» представлення (частина нейронів дає нуль), що також іноді покращує узагальнюючу здатність.

L2-регуляризація на *Dense*-шарах знову ж таки стримує ваги від надмірного росту. *Dropout* з меншими значеннями (0.2 і 0.1) додає трошки шуму в процес навчання, не дозволяючи мережі надто «причепитись» до окремих нейронів.

Вихідний шар це останній *Dense*-шар має стільки нейронів, скільки параметрів передбачається: наприклад, розмір шрифту, рівень контрасту, прапорець спрощеного режиму, вимкнення анімацій тощо. Тут використовується *activation='linear'*, тобто фактично відсутність явної нелінійності. Це означає, що мережа просто видає безпосередні числові значення.

Такий вибір логічний, бо задача — регресія: прогнозуються реальні числа (нормалізовані шрифти й контраст) і бінарні прапорці, які потім можна звести до 0 чи 1 порогом.

На завершення модель компілюється, тобто визначається як модель буде вчитись. Оптимізатор *Adam* обрано тому, що це один із найстабільніших і «безпроблемних» алгоритмів оптимізації для нейромереж. Він поєднує ідеї двох підходів — Momentum та RMSProp. Adam автоматично підлаштовує крок навчання для кожного параметра окремо, добре поводиться на «шумних» градієнтах і зазвичай дає хороші результати без довгого підбору гіперпараметрів.

Для реальних застосунків, особливо коли даних небагато, це практично «дефолтний» безпечний варіант. Функція втрат — mse (mean squared error, середньоквадратична помилка). Вона підходить для регресійних задач, де важливо штрафувати великі помилки сильніше, ніж маленькі: квадрат відхилення росте швидше, тож якщо модель сильно помиляється з деяким користувачем, це помітно впливає на навчання. Як метрики обрано *mae* та *mse*.

MAE (mean absolute error) показує середню абсолютну різницю між передбаченим значенням і фактичним. Це більш «людська» метрика: її легко інтерпретувати як «у середньому модель помиляється на стільки-то одиниць у масштабі нормалізованих параметрів».

MSE використовується і як функція втрат, і як додаткова метрика, щоб бачити, як змінилася квадратична помилка на тренуванні й валідації.

MAE — для інтерпретації, *MSE* — для контролю збіжності й стабільності навчання.

Після компіляції моделі її потрібно “навчити”, для цього використовується функція *train().* Код функції представлений на рисунку 3.14.

```python
def train(self, X_train, y_train, X_val=None, y_val=None):
    """Навчання моделі"""
    if self.model is None:
        self.build_model(X_train.shape[2], y_train.shape[1])

    # Callbacks
    callbacks = [
        EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True),
        ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10, min_lr=1e-7),
        ModelCheckpoint(
            os.path.join(self.config['MODELS_PATH'], 'best_lstm_model.keras'),
            monitor='val_loss',
            save_best_only=True,
            save_weights_only=False
        )
    ]

    validation_data = (X_val, y_val) if X_val is not None else None

    # Навчання
    history = self.model.fit(
        X_train, y_train,
        batch_size=self.config['BATCH_SIZE'],
        epochs=self.config['EPOCHS'],
        validation_data=validation_data,
        callbacks=callbacks,
        verbose=1,
        shuffle=True
    )

    return history
```

Рис. 3.15 Функція для тренування моделі

На самому початку функція дивиться, чи модель вже існує. Якщо це перше навчання — *self.model* ще немає. Тоді функція автоматично викликає *build_model()*, передаючи туди змінні *X_train.shape[2]* — кількість ознак у кожному кроці послідовності та *y_train.shape[1]* — скільки параметрів потрібно передбачити. Якщо модель вже зібрана (наприклад, якщо викликається донавчання) — то виклик побудови моделі пропускається.

Наступний крок налаштування колбеків. Перший колбек *EarlyStopping* – потрібен для спостереження за валідаційною помилкою, якщо модель 15 епох поспіль не покращується — тренування автоматично зупиняється, бо

продовження навчання немає сенсу. Параметр *restore_best_weights=True* означає, що після зупинки повернуться найкращі ваги, отримані під час навчання, а не ті, що були в останній невдалій епосі.

Другий колбек *ReduceLROnPlateau* – якщо модель наближається до оптимального рішення, великий *learning rate* може заважати їй “точно докрутити” ваги зменшення темпу навчання дозволяє моделі тонко донавчитись.

Третій – *ModelCheckpoint* зберігає модель кожного разу, коли вона стала кращою, ніж попередні версії. Параметр *save_best_only=True* означає записувати тільки ту модель, яка показала найнижчу val_loss. В кінцевому результаті маєма найякісніший варіант моделі.

Перш ніж розпочати тренування, функція формує набір даних для валідації, *validation_data* – перевіряє чи є валідаційні дані, якщо є вони підключаються до навчання, якщо ні модель буде тренуватись без валідації.

Фінальний етап запуск навчання моделі методом *fit()* передаються змінні: *X_train, y_train* це тренувальні послідовності й відповідні таргети, *batch_size* розмір партії, визначає скільки послідовностей мережа обробляє одночасно, *epochs* тобто скільки разів модель пройде по всьому набору даних, *validation_data* для перевірка після кожної епохи, *callbacks* для автоматичного контролю зупинки, збереження моделі та регулювання *learning rate*, змінна verbose=1 показує читабельні логи тренування та змінна *shuffle=True* щоб перед кожною епохою дані перемішувались, тим самим зменшуючи шанс переузгодження.

У результаті функція повертає об’єкт history, який містить докладну статистику навчання за кожну епоху: значення помилки, середньої абсолютної похибки, середньоквадратичної похибки та поведінку кривих валідації. Цей об’єкт використовується для аналізу якості моделі, побудови графіків та прийняття рішення щодо подальших кроків оптимізації.

Функція train() є фінальним етапом пайплайну навчання моделі яка відповідає за створення архітектури, контролює перебігу тренування, автоматичну оптимізацію гіперпараметрів та збереження найкращого варіанту

нейронної мережі. У сукупності ці весь пайплайн має за мету побудову точної модель прогнозування поведінки користувача із порушеннями зору.

3.4 Тестування та валідація додатку

На рисунку 3.15 наведено фрагмент логів навчання моделі LSTM, який демонструє поведінку основних метрик під час останніх епох тренування. Скріншот відображає динаміку зміни помилки, точності та параметрів оптимізації, що дозволяє оцінити стабільність процесу навчання та якість узагальнення моделі на валідаційних даних.

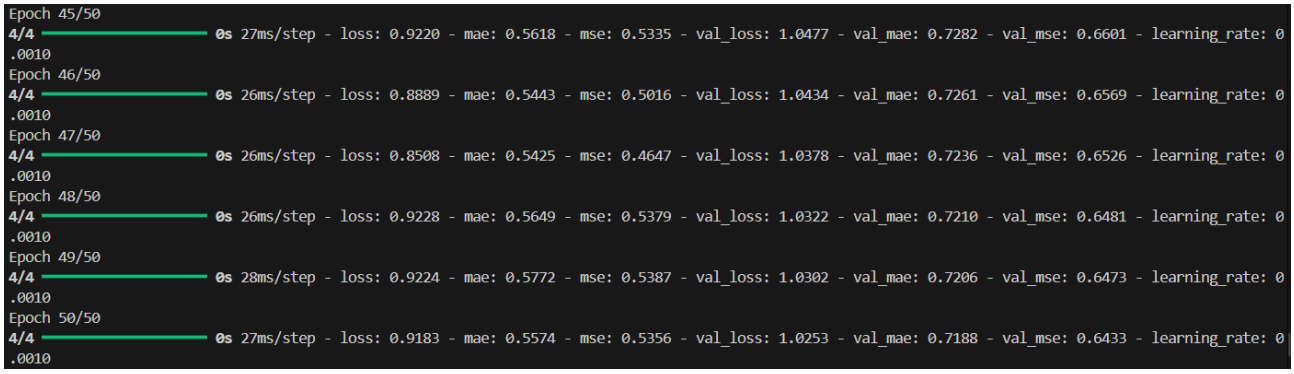


Рис. 3.16 Тренування моделі з 45 по 50 епоху

З рисунку можна зробити висновок, що модель навчається стабільно значення *loss* та *val_loss* плавно змінюються без різких стрибків. Перенавчання не спостерігається адже різниця між *train* і *val* помилками невелика. *MAE* і *MSE* метрики на тренуванні та валідації знаходяться в прийнятних межах і рухаються у правильному напрямку, тому можна сказати, що навчання ефективне. Результати останніх епох майже не відрізняються, це може означати, що модель стабілізувала навчання і знайшла оптимальні параметри. Всі 50 епох пройдено, без збоїв, показники адекватні для задачі прогнозування поведінки користувача, тому навчання завершене коректно.

Наступний рисунок 3.16 показує набори даних реальні *real* та дані які передбачила модель *pred*. Реальні, фактичні значення цільових параметрів

інтерфейсу, які містяться у валідаційному наборі. Передбачені дані - значення, які модель LSTM передбачила на основі поведінкової послідовності користувача.

```
Real: [[ 0.43196454  0.82425696  1.          1.        ]
 [ 0.43196454  0.82425696  1.          1.        ]
 [ 1.24013018  1.26840081  1.          1.        ]
 [ 1.24013018  1.26840081  1.          1.        ]
 [ 1.24013018  1.26840081  1.          1.        ]
 [-1.18436675 -1.69255819  0.          0.        ]
 [-1.18436675 -1.69255819  0.          0.        ]
 [-1.18436675 -1.69255819  0.          0.        ]
 [-1.18436675 -1.69255819  0.          0.        ]
 [-1.18436675 -1.69255819  0.          0.        ]]
Pred: [[ 0.19987103  0.00879483  0.47406456  0.28608876]
 [ 0.1915519   0.00618349  0.4848832   0.2998042 ]
 [ 0.19508961  0.00375472  0.49270785  0.30273056]
 [ 0.25244358  0.02438609  0.4934225   0.32377702]
 [ 0.23921618  0.03822876  0.5088214   0.32843068]
 [ 0.2936175   0.17706549  0.52265114  0.4102284 ]
 [ 0.2591675  -0.00647903  0.53513956  0.383692  ]
 [ 0.03678793 -0.17996672  0.4552959   0.22821352]
 [-0.26368436 -0.27800778  0.33945122  0.10342477]
 [-0.26006973 -0.25701398  0.3287053   0.10679562]]
```

Рис. 3.17 Масив наборів даних реальних і передбачених моделлю

В загальному можна сказати що модель отримує реальні дані, навчається на них і намагається передбачити подібні значення. Результати не є ідеально точними, але модель досить добре вловлює закономірності та напрямок змін. Це свідчить про правильну побудову пайплайну, коректну нормалізацію ознак і здатність моделі відтворювати поведінкові патерни користувача.

На рисунку 3.17 показано підсумкову оцінку роботи моделі на валідаційному наборі даних, тобто на тих даних, яких модель не бачила під час навчання. Це дозволяє зрозуміти, наскільки добре модель узагальнює інформацію та чи здатна робити коректні прогнози в реальних умовах.

```
=== Підсумкова оцінка моделі на валідації ===
val_loss (MSE): 1.0253
val_MAE:        0.7188
val_MSE:        0.6433
```

Рис. 3.18 Підсумкова оцінка моделі на валідації

Показник *val_loss (MSE)* = 1.0253. Це середньоквадратична помилка (Mean Squared Error), яка використовувалась як основна функція втрат. Вона показує, наскільки сильно передбачені значення відрізняються від реальних: чим менше значення тим точніше модель. MSE особливо чутлива до великих помилок, тому її зручніше використовувати для відстеження глобальних відхилень.

Показник *val_MAE* = 0.7188. Це середня абсолютна помилка (Mean Absolute Error). Вона демонструє, наскільки в середньому модель помиляється в кожному передбаченні. Це більш «людська» метрика, адже її легко інтерпретувати: у середньому модель відхиляється приблизно на 0.71 одиниці у нормалізованому масштабі.

Показник *val_MSE* = 0.6433 — дублює основну метрику MSE, але подається окремо як додаткова для контролю. Це показує стабільність результатів і підтверджує, що модель не має різких стрибків у точності.

Ці значення свідчать про те, що модель працює стабільно, адекватно реагує на нові дані та не демонструє ознак перенавчання. Вона здатна прогнозувати поведінкові параметри користувача з прийнятним рівнем точності, що є хорошим результатом для задачі персоналізації інтерфейсу.

Рисунок 3.18 показує як змінювалась помилка моделі (MSE) під час процесу навчання.

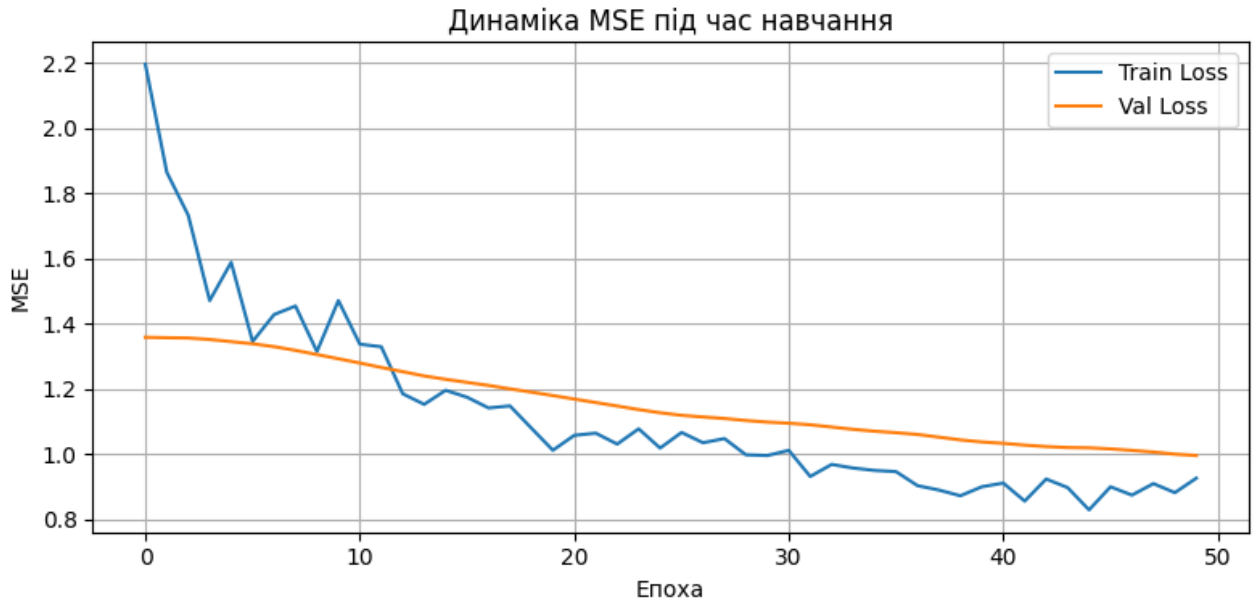


Рис. 3.19 Динаміка MSE під час навчання

Синя лінія відображає помилку на тренувальних даних, а помаранчева — на валідаційних, тобто на нових даних, яких модель не бачила. Спочатку помилка на тренуванні досить висока, і це нормально, адже модель лише починає підлаштовуватися під дані, уже з перших епох видно швидке зменшення Train Loss — модель активно вчиться й знаходить правильні закономірності. Далі крива стабілізується й поступово продовжує знижуватись. Помилка на валідації зменшується повільніше, але рівномірно. Це свідчить про те, що модель не просто запам’ятовує тренувальні дані, а дійсно навчилась узагальнювати інформацію, обидві криві рухаються вниз і не розходяться — це ознака відсутності перенавчання. До кінця навчання обидві метрики зближуються приблизно до значення 1.0, демонструючи стабільну роботу моделі та хорошу точність прогнозування. Графік підтверджує, що вибрана архітектура та процес тренування були ефективними.

Слідуючий графік на рисунку 3.19 показує як змінювалась середня абсолютна помилка (MAE) під час навчання моделі. MAE показує, наскільки в середньому модель помиляється у своїх прогнозах, чим нижче значення — тим краща точність.

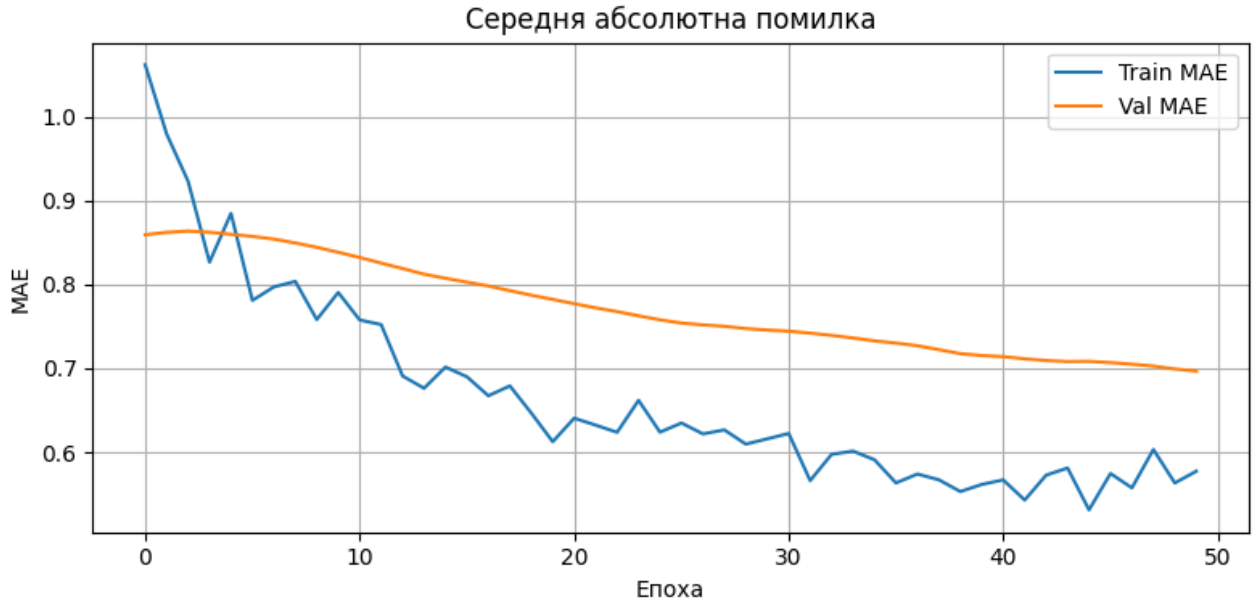


Рис. 3.20 Середня абсолютна помилка

Після запуску оцінки навчання, продуктивність аналізу моделі стабільна, помилка зменшилася як під час навчання, так і на валідаційних даних. Усі епохи та на навчальних даних метрики MSE та MAE зменшилися, тоді як різниця між метриками навчання та валідації залишалася невеликою. Це свідчить про відсутність перенавчання та хороше узагальнення інформації, а не просто процес запам'ятовування. З обраною архітектурою LSTM за правильної конфігурації та правильного процесу нормалізації, формування послідовностей та навчання мережі, динаміка навчання загалом підтверджує потужність обраної архітектури. Модель демонструє достатню точність для задачі прогнозування поведінкових параметрів користувача та може використовуватись у системі адаптації інтерфейсу.

ВИСНОВКИ

1. Проведено аналіз сучасних рішень у сфері адаптивних інтерфейсів, орієнтованих на людей з обмеженими можливостями. Встановлено, що більшість існуючих інструментів залишаються статичними, зосередженими переважно на зміні контенту, а не на адаптації взаємодії, що обмежує їхню ефективність у реальних сценаріях використання.

2. Проаналізовано міжнародні стандарти доступності, зокрема ISO 9241, WCAG 2.2, EN 301 549 та ADA. Ці стандарти формують нормативний фундамент, але вони не завжди охоплюють динамічні поведінкові особливості користувачів і не передбачають механізмів адаптації в реальному часі.

3. Обґрунтовано застосування рекурентних нейронних мереж та LSTM для моделювання персоналізованої взаємодії користувача з цифровим середовищем. Такий підхід дав змогу створити механізм прогнозування, який не лише реагує на виконувані дії, а й передбачає подальші потреби користувача, забезпечуючи вищий рівень персоналізації.

4. Виявлено обмеження існуючих браузерних розширень, що пропонують допомогу людям із порушеннями зору. Проведене порівняння показало, що ці інструменти не володіють здатністю до самоадаптації та потребують ручного налаштування, тоді як система на основі RNN усуває ці недоліки завдяки автоматичному прогнозуванню та корекції параметрів інтерфейсу.

5. Проведено моделювання та тренування нейронної мережі, яка продемонструвала достатню точність для прогнозування поведінкових параметрів користувача. Отримані результати підтвердили можливість інтеграції моделі як активного компонента системи автоматизованої адаптації інтерфейсу.

6. Реалізовано та протестовано браузерне розширення, здатне автоматично змінювати параметри вебінтерфейсу відповідно до індивідуальних потреб користувачів із порушеннями зору. Тестування підтвердило життєздатність запропонованого підходу: система коректно адаптує вебвміст без необхідності

ручного налаштування кожного сайту, тим самим підвищуючи доступність та комфорт користування.

Робота пройшла апробацію на конференціях та опубліковано у наступних тезах:

1. Ярощук Б.О., Герцюк М.М. ПЕРСОНАЛІЗОВАНІ АДАПТИВНІ ІНТЕРФЕЙСИ ДЛЯ ЛЮДЕЙ З ОБМЕЖЕНИМИ МОЖЛИВОСТЯМИ. II Міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії», 19-21 грудня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.236.
2. Ярощук Б.О., Шахматов І.О. АЛГОРИТМ РОБОТИ СИСТЕМИ АДАПТАЦІЇ ІНТЕРФЕЙСУ ДЛЯ КОРИСТУВАЧІВ ІЗ ОБМЕЖЕНИМИ МОЖЛИВОСТЯМИ. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.16.

ПЕРЕЛІК ПОСИЛАНЬ

1. Digital 2023: Global Overview Report: Режим доступу до ресурсу: [Електронний ресурс]. – Режим доступу до ресурсу: https://datareportal.com/reports/digital-2023-global-overview-report

2. World health organization [Електронний ресурс]. – Режим доступу до ресурсу: https://www.who.int/news-room/fact-sheets/detail/disability-and-health

3. Міністерство соціальної політики України. Особам з інвалідністю [Електронний ресурс]. – Режим доступу до ресурсу: https://www.msp.gov.ua/timeline/invalidnist.html

4. Марія Олійник. Київський національний економічний університет імені Вадима Гетьмана, “Глобальна проблема цифрової нерівності та її ключові форми.” [Електронний ресурс]. – Режим доступу до ресурсу: https://economyandsociety.in.ua/index.php/journal/article/view/3136/3059

5. Адаптивні користувацькі інтерфейси: роль машинного навчання у персоналізації [Електронний ресурс]. – Режим доступу до ресурсу: https://peerdh.com/uk/blogs/programming-insights/adaptive-user-interfaces-the-role-of-machine-learning-in-personalization

6. Визначення слова Інтерфейс [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Інтерфейс

7. Celebrating Penn Engineering History: ENIAC [Електронний ресурс]. – Режим доступу до ресурсу: https://www.seas.upenn.edu/about/history-heritage/eniac/

8. The Mother of All Demos, presented by Douglas Engelbart (1968). [Електронний ресурс]. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=yJDv-zdhzMY

9. Schlungbaum, E. Individual User Interfaces and Model-Based User Interface Software Tools. In Proceedings of the 2nd International Conference on Intelligent User

Interfaces, Orlando, FL, USA, 6–9 January 1997; Association for Computing Machinery: New York, NY, USA, 1997; pp. 229–232. [Електронний ресурс]. – Режим доступу до ресурсу: https://dl.acm.org/doi/pdf/10.1145/238218.238330

10. Yutan Huang, Tanjila Kanij, Anuradha Madugalla, Shruti Mahajan, Chetan Arora, John Grundy “Unlocking Adaptive User Experience with Generative AI” [Електронний ресурс]. – Режим доступу до ресурсу: https://arxiv.org/abs/2404.05442

11. Angela Carrera-Rivera, Daniel Reguera-Bakhache, Felix Larrinaga, Ganix Lasa “Exploring the transformation of user interactions to Adaptive Human-Machine Interfaces” [Електронний ресурс]. – Режим доступу до ресурсу: https://arxiv.org/abs/2311.03806

12. Saša Brdnik, Tjaša Heričko, Boštjan Šumak “Intelligent User Interfaces and Their Evaluation: A Systematic Mapping Study” [Електронний ресурс]. – Режим доступу до ресурсу: https://www.mdpi.com/1424-8220/22/15/5830

13. R.Cowie, E. Douglas-Cowie, N. Tsapatsolius, G. Votsis, S. Kollias, W. Fellenz and J.G. Taylor “Emotion recognition in human computer interaction” [Електронний ресурс]. – Режим доступу до ресурсу: https://www.researchgate.net/publication/3321357_Emotion_recognition_in_human-computer_interaction

14. Yuanyuan Xu, Yin-Shan Lin, Xiaofan Zhou, Xinyang Shan, 2024; “Utilizing emotion recognition technology to enhance user experience in real-time” [Електронний ресурс]. – Режим доступу до ресурсу: https://www.semanticscholar.org/paper/Utilizing-emotion-recognition-technology-to-enhance-Xu-Lin/db79111111c45b0a90cf9f6942938b88e0890f79

15. Helma Torkamaan, Mohammad Tahaei, Stefan Buijsman, Ziang Xiao, Daricia Wilkinson & Bart P. Knijnenburg “The Role of Human-Centered AI in User Modeling, Adaptation, and Personalization—Models, Frameworks, and Paradigms”

[Електронний ресурс]. – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/978-3-031-55109-3_2

16.Gerhard Fischer, “Adaptive and Adaptable Systems: Differentiating and Integrating AI and EUD” [Електронний ресурс]. – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/978-3-031-34433-6_1

17. Lamia Zouhaier, Yousra BenDalyHlaoui & Leila Ben Ayed, “Adaptive user interface based on accessibility context” [Електронний ресурс]. – Режим доступу до ресурсу: https://link.springer.com/article/10.1007/s11042-023-14390-5

18. Iqbal H. Sarker, Alan Colman, Jun Han, and Paul Watters “Context-Aware Machine Learning and Mobile Data Analytics: Automated Rule-based Services with Intelligent Decision-Making” [Електронний ресурс]. – Режим доступу до ресурсу: https://link.springer.com/book/10.1007/978-3-030-88530-4

19. Kashyap Todi, Gilles Bailly, Luis A. Leiva, and Antti Oulasvirta, “Adapting User Interfaces with Model-based Reinforcement Learning” [Електронний ресурс]. – Режим доступу до ресурсу: https://dl.acm.org/doi/fullHtml/10.1145/3411764.3445497

20. Jamil Hussain, Anees Ul Hassan, Hafiz Syed Muhammad Bilal, Rahman Ali, Muhammad Afzal, Shujaat Hussain, Jaehun Bang, Oresti Banos, and Sungyoung Lee, "Model-based adaptive user interface based on context and user experience" [Електронний ресурс]. – Режим доступу до ресурсу: https://link.springer.com/article/10.1007/s12193-018-0258-2

21. Jamil Hussain, Anees Ul Hassan, Hafiz Syed Muhammad Bilal, Rahman Ali, Muhammad Afzal, Shujaat Hussain, Jaehun Bang, Oresti Banos, and Sungyoung Lee, "Design, Development, and Deployment of Context-Adaptive AI Systems for Human-Centered Interaction" [Електронний ресурс]. –Режим доступу до ресурсу: https://link.springer.com/article/10.1007/s12193-018-0258-2

22. Yusei Ishimizu, Jialong Li, Jinglue Xu, Jinyu Cai, Hitoshi Iba, Kenji Tei, “Automatic Adaptation Rule Optimization via Large Language Models” [Електронний ресурс]. – Режим доступу до ресурсу: https://arxiv.org/abs/2407.02203

23. ISO 9241-171:2008(en), Ergonomics of human-system interaction [Електронний ресурс]. – Режим доступу до ресурсу: https://www.iso.org/obp/ui/en/#iso:std:iso:9241:-171:ed-1:v1:en

24. ISO 9241-11:2018(en) Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts [Електронний ресурс]. – Режим доступу до ресурсу: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en

25. ISO/IEC 25010:2023(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model [Електронний ресурс]. – Режим доступу до ресурсу: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:25010:ed-2:v1:en

26. EN 301 549 V3.2.1 (2021-03) Accessibility requirements for ICT products and services [Електронний ресурс]. – Режим доступу до ресурсу: https://www.etsi.org/deliver/etsi_en/301500_301599/301549/03.02.01_60/en_301549v030201p.pdf

27. The Americans with Disabilities Act (ADA) [Електронний ресурс]. – Режим доступу до ресурсу: https://www.ada.gov/resources

28. Web Content Accessibility Guidelines (WCAG) 2.1 [Електронний ресурс]. – Режим доступу до ресурсу: https://www.w3.org/TR/WCAG21/#abstract

29. ADA compliance та стандарт WCAG: як застосовувати компаніям? [Електронний ресурс]. – Режим доступу до ресурсу: https://biz.ligazakon.net/analitycs/227164_ada-sompliance-ta-standart-wcag-yak-zastosovuvati-kompanyam.

30. What is k-means clustering? [Електронний ресурс]. – Режим доступу: https://www.ibm.com/think/topics/k-means-clustering

31. Cao, J., Lam, K.-Y., Lee, L.-H., Liu, X., Hui, P., Su, X. Mobile Augmented Reality: User Interfaces, Frameworks, and Intelligence [Електронний ресурс] // ACM Computing Surveys. – 2023. – Vol. 55, Issue 9. – Article No. 189. – С. 1–36. – Режим доступу: https://doi.org/10.1145/3557999.

32. García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., Herrera, F. Big data preprocessing: methods and prospects [Електронний ресурс] // Big Data Analytics. – 2016. – Vol. 1, Issue 9. – Режим доступу: https://doi.org/10.1186/s41044-016-0014-0.

33. Edwards, E. J., Donoghue, M. J. Is it easy to move and easy to evolve? Evolutionary accessibility and adaptation [Електронний ресурс] // Journal of Experimental Botany. – 2013. – Vol. 64, Issue 13. – P. 4047–4052. – Режим доступу: https://doi.org/10.1093/jxb/ert220

34. Hussain, J., Hassan, A. U., Bilal, H. S. M., Ali, R., Afzal, M., Hussain, S., Bang, J., Banos, O., Lee, S. Model-based adaptive user interface based on context and user experience evaluation [Електронний ресурс] // Journal on Multimodal User Interfaces. – 2018. – Vol. 12, Issue 10. – Режим доступу: https://doi.org/10.1007/s12193-018-0258-2

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська Робота

«Автоматизація адаптації інтерфейсу під потреби користувача із обмеженими можливостями»

Виконав: студент групи ПДМ -61 Богдан ЯРОЩУК

Керівник: канд. техн. наук, доцент кафедри ІПЗ Тимур ДОВЖЕНКО

Київ - 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення автоматизованої адаптації інтерфейсів за допомогою штучного інтелекту, для забезпечення інклюзії людей з обмеженими можливостями.

Об’єкт дослідження: процес адаптації інтерфейсів цифрових продуктів, які використовуються людьми з обмеженими можливостями.

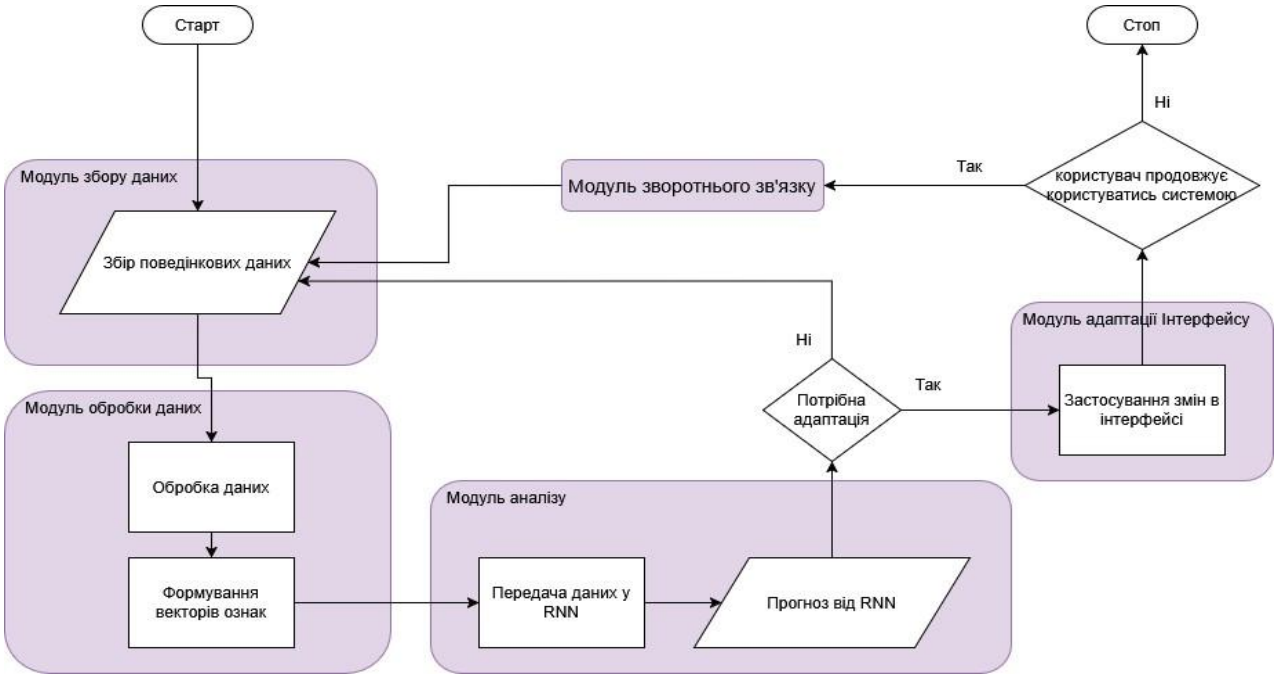
Предмет дослідження: методи та моделі застосування технологій штучного інтелекту для створення інтерфейсів, що адаптуються під індивідуальні потреби людей з обмеженими можливостями.

АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО АДАПТАЦІЇ

Підхід	Як працює	Недолік	Переваги запропонованої системи
Адаптації засновані на правилах	Фіксовані правила змінюють шрифти, кольори, елементи керування тощо	Статичність, не реагують на поведінку, не персоналізують, реагують тільки по правилах	Динамічна адаптація залежно від реальних дій користувача
Контекстні адаптації	Реагують на пристрій, розмір екрана, умови середовища тощо	Не враховують індивідуальні потреби людей з обмеженнями, не враховують специфічний контекст	Поведінкова аналітика, персоналізація для кожного користувача
Стандартні засоби доступності	Користувач сам змінює параметри	Вимагає ручного налаштування, не працює автоматично	Автоматична адаптація без участі користувача
Спеціалізовані рішення на рівні платформи	Залежність від розробника	Не універсальні, залежать від платформи	Універсальність яка може бути інтегрована у будь -яку платформу
Існуючі AI-рішення доступності	Локальні сценарії персоналізації	Обмежена гнучкість, відсутність моделювання поведінки в часі	Моделювання послідовностей через RNN, глибша індивідуалізація

3

КОНЦЕПТУАЛЬНА МОДЕЛЬ ТА АЛГОРИТМ СИСТЕМИ



4

ЗБІР ДАНИХ КОРИСТУВАЧА

5

ВХІДНА ПОСЛІДОВНІСТЬ ВЕКТОРІВ СТАНУ

За одну сессію дії користувача представляються як послідовність векторів стану:

$$X = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10} \dots, x_T)$$

x_1 - розмір шрифту,
x_2 - рівень контрасту,
x_3 - міжрядковий інтервал
x_4 - відстань між літерами
x_5 - відстань між словами
x_6 - використаний шрифт
x_7 - активна тема
x_8 - масштаб сторінки
x_9 - контроль анімацій
x_{10} - автоматичне відображення альтернативного тексту

6

МЕХАНІЗМ ОНОВЛЕННЯ ПАМ'ЯТІ В LSTM

Перший LSTM Шар

1. Вентиль Забування (f_t): Визначає, що забути зі старого стану комірки ct−1.

$$f_t = \sigma(W_f h_{t-1} + W_f x_t + b_f)$$

2. Вхідний Вентиль (i_t): Визначає, що додати до стану комірки.

$$i_t = (W_i h_{t-1} + W_i x_t + b_i)$$

3. Потенційний Стан ($\widetilde{C}_t$): Нова інформація

$$\tilde{C}_t = \tanh(W_C h_{t-1} + W_C x_t + b_C)$$

Вихід шару: шар виводить повну послідовність прихованих станів

$$H^{(1)} = (h_1, h_2, \dots, h_T)$$

7

ОНОВЛЕННЯ СТАНУ ТА МЕХАНІЗМИ СТАБІЛІЗАЦІЇ

4. **Оновлення Стану Комірки (c_t):** Забування старого та додавання нової інформації.

$$c_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

5. **Прихований Стан (h_t):** Вихідний вентиль (o_t) визначає, що вивести.

$$o_t = \sigma(W_o h_{t-1} + W_o x_t + b_o)$$

$$h_t = o_t \odot \tanh(c_t)$$

Другий LSTM Шар та Регуляризація

- Другий LSTM (64 одиниці): Повторює обчислення вентилів, шар виводить лише фінальний вектор прихованого стану h_T.
- **Batch Normalization:** Застосовується після обох шарів LSTM
- **Dropout (0.3):** Випадкове обнулення 30% елементів на виході LSTM

8

ТРАНСФОРМАЦІЯ ОЗНАК ТА ФІНАЛЬНА РЕГРЕСІЯ

Dense Шари

• **Лінійне перетворення:**

$$Z^{(L)} = A^{(L-1)}W^{(L)} + b^{(L)}$$

• **Активація ReLU:**

$$A^{(L)} = \text{ReLU}(Z^{(L)}) = \max(0, Z^{(L)})$$

Регуляризація Dense Шару

• **Batch Normalization:** Застосовується після Dense(32)

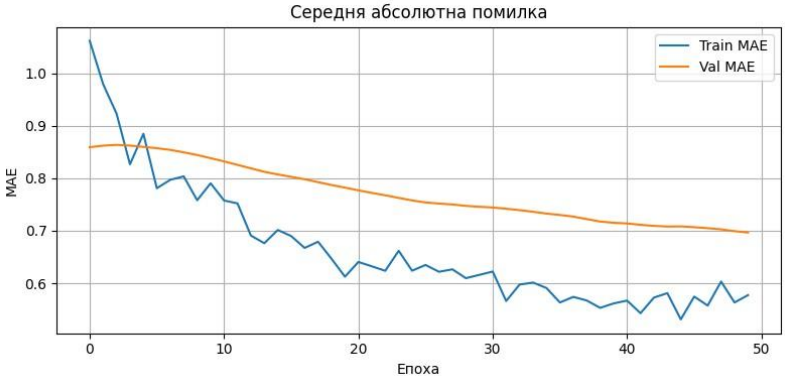
• **Dropout:** Застосовується зі зменшенням інтенсивності (0.2, потім 0.1)

Вихідний Шар: Виконується останнє лінійне перетворення для отримання прогнозу ($\hat{y}$).

$$\hat{y} = A_{final}\ W_{out} + b_{out}$$

9

РЕЗУЛЬТАТИ НАВЧАННЯ МОДЕЛІ LSTM



ВИСНОВКИ

1. Проведено аналіз сучасних рішень у сфері адаптивних інтерфейсів, орієнтованих на людей з обмеженими можливостями.

2. Проаналізовано існуючі стандарти доступності (ISO 9241, WCAG 2.2, EN 301 549, ADA).

3. Проведено аналіз рекурентних нейронних мереж та їх версії LTSM для персоналізації взаємодії з цифровим середовищем.

4. Сформовано концепцію адаптивного інтерфейсу, що ґрунтується на динамічному врахуванні поведінкових характеристик, контексту використання та специфічних обмежень користувачів із порушенням зору.

5. Проведене тестування шляхом розробки браузерного розширення, що автоматично змінює параметри веб-інтерфейсів залежно від індивідуальних особливостей користувача із порушенням зору.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Ярощук Б.О., Герцюк М.М. ПЕРСОНАЛІЗОВАНІ АДАПТИВНІ ІНТЕРФЕЙСИ ДЛЯ ЛЮДЕЙ З ОБМЕЖЕНИМИ МОЖЛИВОСТЯМИ. ІІ Міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії», 19-21 грудня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.236.
2. Ярощук Б.О., Шахматов І.О. АЛГОРИТМ РОБОТИ СИСТЕМИ АДАПТАЦІЇ ІНТЕРФЕЙСУ ДЛЯ КОРИСТУВАЧІВ ІЗ ОБМЕЖЕНИМИ МОЖЛИВОСТЯМИ. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.16.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Manifest.json

```
{
 "manifest_version": 3,
 "name": "Accessibility Adaptor",
 "version": "1.0.0",
 "description": "Адаптація інтерфейсу відповідно до WCAG 2.2 та ISO 9241-171",

 "permissions": [
  "activeTab",
  "storage",
  "scripting",
  "webNavigation",
  "tabs"
 ],

 "host_permissions": [
  "<all_urls>"
 ],

 "background": {
  "service_worker": "background/background.js"
 },

 "content_scripts": [
  {
   "matches": ["<all_urls>"],
   "js": [
    "shared/utils.js",
    "shared/storage-manager.js",
    "content/context-detector.js",
    "content/behavior-tracker.js",
    "content/wcag-adapter.js",
    "content/iso-adapter.js",
    "content/content-ui.js",
    "content/floating-panel.js",
    "content/content.js"
   ],
   "css": ["content/content.css"],
   "run_at": "document_end"
  }
 ],

 "action": {
  "default_popup": "popup/popup.html",
  "default_title": "Налаштування доступності",
  "default_icon": {
  }
 },

 "options_page": "options/options.html",

 "web_accessible_resources": [{
  "resources": ["content/*", "shared/*"],
  "matches": ["<all_urls>"]
 }]
}
```

Popup.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>Accessibility Adaptor</title>
   <link rel="stylesheet" href="popup.css">
```

```
</head>
<body>
  <div class="accessibility-popup">
    <header class="popup-header">
      <h1>🎯 Доступність</h1>
      <div class="status-indicator">
        <span class="status-dot active"></span>
        <span class="status-text">Активний</span>
      </div>
    </header>

    <!-- Швидкі перемикачі -->
    <div class="quick-toggles">
      <button class="toggle-btn" data-feature="simplified-view">
        <i class="icon">👁️</i>
        <span>Спрощений вид</span>
      </button>

      <button class="toggle-btn" data-feature="high-contrast">
        <i class="icon">⚫⚪</i>
        <span>Високий контраст</span>
      </button>

      <button class="toggle-btn" data-feature="large-targets">
        <i class="icon">🎯</i>
        <span>Великі кнопки</span>
      </button>

      <button class="toggle-btn" data-feature="no-animation">
        <i class="icon">⏸️</i>
        <span>Стоп анімації</span>
      </button>
    </div>

    <!-- Контекстні пресети -->
    <div class="context-section">
      <label for="context-selector">Режим:</label>
      <select id="context-selector">
        <option value="auto">🔄 Автоматично</option>
        <option value="reading">📖 Читання</option>
        <option value="forms">📝 Форми</option>
        <option value="video">🎬 Відео</option>
        <option value="shopping">🛒 Шопінг</option>
      </select>
    </div>

    <!-- Рівні адаптації -->
    <div class="level-section">
      <label>Рівень адаптації:</label>
      <div class="level-selector">
        <button class="level-btn active" data-level="A">A</button>
        <button class="level-btn" data-level="AA">AA</button>
        <button class="level-btn" data-level="AAA">AAA</button>
      </div>
    </div>

    <!-- AI Рекомендації -->
    <div class="ai-recommendation" id="ai-recommendation">
      <div class="ai-header">
        <span class="ai-icon">🤖</span>
        <strong>AI пропонує:</strong>
      </div>
      <div class="ai-suggestion" id="ai-suggestion">
        Застосувати великий текст для читання
      </div>
      <button class="apply-ai-btn" id="apply-ai">Застосувати</button>
    </div>
```

```
        <div class="popup-actions">
            <button id="apply-settings">✅ Застосувати</button>
            <button id="advanced-settings">⚙️ Розширені налаштування</button>
            <button id="toggle-extension">🔴 Вимкнути розширення</button>
        </div>
    </div>
    <script src="popup.js"></script>
</body>
</html>
```

Popup.js

```
class PopupManager {
    constructor() {
        this.currentSettings = {};
        this.isEnabled = true;
        this.init();
    }

    async init() {
        await this.loadSettings();
        this.setupEventListeners();
        this.updateUI();
    }

    async loadSettings() {
        const result = await chrome.storage.local.get([
            'userSettings',
            'isEnabled',
            'currentLevel',
            'aiRecommendation'
        ]);

        this.currentSettings = result.userSettings || {};
        this.isEnabled = result.isEnabled !== false;
        this.currentLevel = result.currentLevel || 'A';
        this.aiRecommendation = result.aiRecommendation;
    }

    setupEventListeners() {
        // Перемикачі функцій
        document.querySelectorAll('.toggle-btn').forEach(btn => {
            btn.addEventListener('click', (e) => this.toggleFeature(e));
        });

        // Рівні адаптації
        document.querySelectorAll('.level-btn').forEach(btn => {
            btn.addEventListener('click', (e) => this.changeLevel(e));
        });

        // Контекстний селектор
        document.getElementById('context-selector').addEventListener('change', (e) => {
            this.changeContext(e.target.value);
        });

        // AI рекомендації
        document.getElementById('apply-ai').addEventListener('click', () => {
            this.applyAIRecommendation();
        });

        // Кнопки дій
        document.getElementById('advanced-settings').addEventListener('click', () => {
            chrome.runtime.openOptionsPage();
        });

        document.getElementById('toggle-extension').addEventListener('click', () => {
            this.toggleExtension();
        });
```

```
    document.getElementById('apply-settings').addEventListener('click', async () => {
        // Отримуємо поточні налаштування з UI
        const [tab] = await chrome.tabs.query({ active: true, currentWindow: true });
        console.log('Applying settings to tab:', tab.id, this.currentSettings);
        chrome.tabs.sendMessage(tab.id, {
            action: 'updateBehaviorData',
            settings: this.currentSettings
        });

        // Закриваємо popup
        window.close();
    });
}

async toggleFeature(event) {
    const button = event.currentTarget;
    const feature = button.dataset.feature;
    const isActive = button.classList.contains('active');

    // Оновлюємо UI
    button.classList.toggle('active');

    // Оновлюємо налаштування
    this.currentSettings[feature] = !isActive;
    await this.saveSettings();

    // Відправляємо повідомлення на сторінку
    const [tab] = await chrome.tabs.query({ active: true, currentWindow: true });
    chrome.tabs.sendMessage(tab.id, {
        action: 'updateSettings',
        settings: this.currentSettings
    });
}

async changeLevel(event) {
    const button = event.currentTarget;
    const level = button.dataset.level;

    // Оновлюємо UI
    document.querySelectorAll('.level-btn').forEach(btn => {
        btn.classList.remove('active');
    });
    button.classList.add('active');

    // Зберігаємо рівень
    this.currentLevel = level;
    await chrome.storage.local.set({ currentLevel: level });

    // Застосовуємо налаштування рівня
    const [tab] = await chrome.tabs.query({ active: true, currentWindow: true });
    chrome.tabs.sendMessage(tab.id, {
        action: 'changeLevel',
        level: level
    });
}

async changeContext(context) {
    await chrome.storage.local.set({ currentContext: context });

    const [tab] = await chrome.tabs.query({ active: true, currentWindow: true });
    chrome.tabs.sendMessage(tab.id, {
        action: 'changeContext',
        context: context
    });
}

async applyAIRecommendation() {
    if (!this.aiRecommendation) return;
```

```
        const [tab] = await chrome.tabs.query({ active: true, currentWindow: true });
        chrome.tabs.sendMessage(tab.id, {
            action: 'applyAIRecommendation',
            recommendation: this.aiRecommendation
        });

        // Ховаємо рекомендацію після застосування
        document.getElementById('ai-recommendation').style.display = 'none';
    }

    async toggleExtension() {
        this.isEnabled = !this.isEnabled;
        await chrome.storage.local.set({ isEnabled: this.isEnabled });

        const iconPath = this.isEnabled ? 'icons/icon-16.png' : 'icons/icon-disabled.png';
        chrome.action.setIcon({ path: iconPath });

        this.updateUI();
    }

    async saveSettings() {
        await chrome.storage.local.set({
            userSettings: this.currentSettings
        });
    }

    updateUI() {
        // Оновлюємо стан перемикачів
        Object.entries(this.currentSettings).forEach(([feature, isActive]) => {
            const button = document.querySelector(`[data-feature="${feature}"]`);
            if (button) {
                button.classList.toggle('active', isActive);
            }
        });

        // Оновлюємо рівень
        document.querySelectorAll('.level-btn').forEach(btn => {
            btn.classList.toggle('active', btn.dataset.level === this.currentLevel);
        });

        // Оновлюємо статус
        const statusDot = document.querySelector('.status-dot');
        const statusText = document.querySelector('.status-text');
        const toggleBtn = document.getElementById('toggle-extension');

        if (this.isEnabled) {
            statusDot.className = 'status-dot active';
            statusText.textContent = 'Активний';
            toggleBtn.textContent = '🔴 Вимкнути розширення';
        } else {
            statusDot.className = 'status-dot inactive';
            statusText.textContent = 'Неактивний';
            toggleBtn.textContent = '🟢 Увімкнути розширення';
        }

        // Оновлюємо AI рекомендації
        if (this.aiRecommendation) {
            document.getElementById('ai-suggestion').textContent = this.aiRecommendation.text;
            document.getElementById('ai-recommendation').style.display = 'block';
        } else {
            document.getElementById('ai-recommendation').style.display = 'none';
        }
    }
}

// Ініціалізація при завантаженні
document.addEventListener('DOMContentLoaded', () => {
    new PopupManager();
});
```

Background.js

```
// Фоновий сервісний worker для розширення
class BackgroundService {
    constructor() {
        this.isInitialized = false;
        this.userSessions = new Map();
        this.init();
    }

    async init() {
        try {
            await this.setupMessageHandlers();
            await this.setupNavigationHandlers();
            await this.initializeAI();
            this.isInitialized = true;

            console.log('Accessibility Adaptor: Background service initialized');
        } catch (error) {
            console.error('Accessibility Adaptor: Background init error', error);
        }
    }

    setupMessageHandlers() {
        chrome.runtime.onMessage.addListener((request, sender, sendResponse) => {
            this.handleBackgroundMessage(request, sender, sendResponse);
            return true;
        });

        // Обробка повідомлень між вкладками
        chrome.runtime.onConnect.addListener((port) => {
            this.handlePortConnection(port);
        });
    }

    setupNavigationHandlers() {
        // Відстеження навігації між сторінками
        chrome.webNavigation.onCompleted.addListener((details) => {
            this.handlePageLoad(details);
        });

        chrome.tabs.onActivated.addListener((activeInfo) => {
            this.handleTabSwitch(activeInfo);
        });
    }

    async handleBackgroundMessage(request, sender, sendResponse) {
        const { action, data } = request;

        try {
            switch (action) {
                case 'getUserProfile':
                    const profile = await this.getUserProfile(sender.tab.id);
                    sendResponse({ success: true, profile });
                    break;

                case 'updateBehaviorData':
                    await this.updateBehaviorData(sender.tab.id, data);
                    sendResponse({ success: true });
                    break;

                case 'getAIRecommendation':
                    const recommendation = await this.getAIRecommendation(sender.tab.id);
                    sendResponse({ success: true, recommendation });
                    break;

                case 'syncSettings':
                    await this.syncSettings(data);
                    sendResponse({ success: true });
```

```
                break;

            default:
                sendResponse({ success: false, error: 'Unknown action' });
        }
    } catch (error) {
        sendResponse({ success: false, error: error.message });
    }
}

async getUserProfile(tabId) {
    const session = this.getUserSession(tabId);
    return session.profile;
}

async updateBehaviorData(tabId, behaviorData) {
    const session = this.getUserSession(tabId);
    session.behaviorData.push({
        timestamp: Date.now(),
        ...behaviorData
    });

    // Періодична відправка даних на сервер
    if (session.behaviorData.length >= 10) {
        await this.sendDataToServer(session.behaviorData);
        session.behaviorData = [];
    }
}

async getAIRecommendation(tabId) {
    const session = this.getUserSession(tabId);
    const recentBehavior = session.behaviorData.slice(-5);

    // Тимчасова базова логіка рекомендацій
    return this.generateBasicRecommendation(recentBehavior);
}

generateBasicRecommendation(behavior) {
    // Аналіз останньої поведінки для генерації рекомендацій
    const textZoomCount = behavior.filter(b => b.fontSize > 100).length;
    const contrastChanges = behavior.filter(b => b.contrastMode !== 'normal').length;

    if (textZoomCount > 2) {
        return {
            name: 'Автоматичне збільшення тексту',
            settings: { fontSize: 125 },
            confidence: 0.8
        };
    }

    if (contrastChanges > 1) {
        return {
            name: 'Постійний високий контраст',
            settings: { contrastMode: 'high' },
            confidence: 0.7
        };
    }

    return null;
}

getUserSession(tabId) {
    if (!this.userSessions.has(tabId)) {
        this.userSessions.set(tabId, {
            tabId,
            profile: {},
            behaviorData: [],
            startTime: Date.now()
        });
```

```
    }
    return this.userSessions.get(tabId);
}

async sendDataToServer(data) {
    try {
        const response = await fetch('http://localhost:5000/api/behavior', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({
                data: data,
                sessionId: this.generateSessionId(),
                timestamp: Date.now()
            })
        });

        if (!response.ok) {
            throw new Error('Server response not OK');
        }

        return await response.json();
    } catch (error) {
        console.warn('Failed to send data to server, storing locally');
        await this.storeDataLocally(data);
    }
}

async storeDataLocally(data) {
    const result = await chrome.storage.local.get(['pendingData']);
    const pendingData = result.pendingData || [];
    pendingData.push(...data);

    await chrome.storage.local.set({ pendingData });
}

generateSessionId() {
    return 'session_' + Math.random().toString(36).substr(2, 9);
}

handlePageLoad(details) {
    if (details.frameId === 0) { // Тільки основна frame
        const session = this.getUserSession(details.tabId);
        session.currentUrl = details.url;
        session.pageLoadTime = Date.now();

        // Відправка інформації про завантаження сторінки
        this.sendPageLoadAnalytics(details);
    }
}

handleTabSwitch(activeInfo) {
    // Оновлення активної сесії
    this.userSessions.forEach((session, tabId) => {
        session.isActive = tabId === activeInfo.tabId;
    });
}

async sendPageLoadAnalytics(details) {
    // Відправка анонімної статистики про завантаження сторінки
    const analyticsData = {
        url: details.url,
        loadTime: details.timeStamp,
        tabId: details.tabId
    };

    // Тут може бути додаткова логіка для аналітики
}
```

```
    async initializeAI() {
        console.log('AI system initialized');
    }

async handleBackgroundMessage(request, sender, sendResponse) {
    const { action, data } = request;

    try {
        switch (action) {
            case 'ping':
                // Перевірка доступності
                sendResponse({ success: true, timestamp: Date.now() });
                break;

            case 'getUserProfile':
                const profile = await this.getUserProfile(sender.tab.id);
                sendResponse({ success: true, profile });
                break;

            case 'updateBehaviorData':
                await this.updateBehaviorData(sender.tab.id, data);
                sendResponse({ success: true });
                break;

            case 'getAIRecommendation':
                const recommendation = await this.getAIRecommendation(sender.tab.id);
                sendResponse({ success: true, recommendation });
                break;

            case 'syncSettings':
                await this.syncSettings(data);
                sendResponse({ success: true });
                break;

            case 'settingsApplied':
                // Логуємо застосування налаштувань
                console.log('Settings applied:', data.settings);
                sendResponse({ success: true });
                break;

            default:
                sendResponse({ success: false, error: 'Unknown action' });
        }
    } catch (error) {
        console.error('Background message error:', error);
        sendResponse({ success: false, error: error.message });
    }
}
}

// Ініціалізація сервісу при завантаженні
const backgroundService = new BackgroundService();

// Обробка встановлення розширення
chrome.runtime.onInstalled.addListener((details) => {
    if (details.reason === 'install') {
        // Перша установка - відкриваємо сторінку налаштувань
        chrome.runtime.openOptionsPage();

        // Зберігаємо стандартні налаштування
        chrome.storage.local.set({
            userSettings: StorageManager.getDefaultSettings(),
            isEnabled: true,
            currentLevel: 'A'
        });
    }
});
```

```
// Обробка оновлення розширення
chrome.runtime.onUpdateAvailable.addListener(() => {
  chrome.runtime.reload();
});
```

Content.js

```
// Оновлений контент-скрипт з покращеним механізмом застосування налаштувань
class ContentScript {
  constructor() {
    this.adapters = new Map();
    this.isEnabled = true;
    this.currentSettings = {};
    this.pendingChanges = new Set();
    this.init();
  }

  async init() {
    try {
      await this.loadSettings();
      this.initAdapters();
      this.initUI();
      this.setupMessageListeners();
      this.startBehaviorTracking();
      this.applyAllAdaptations(); // Застосовуємо налаштування при завантаженні

      console.log('Accessibility Adaptor: Ініціалізація завершена');
    } catch (error) {
      console.error('Accessibility Adaptor: Помилка ініціалізації', error);
    }
  }

  async loadSettings() {
    try {
      const result = await chrome.storage.local.get([
        'userSettings',
        'isEnabled',
        'currentLevel',
        'userProfile'
      ]);

      this.currentSettings = result.userSettings || this.getDefaultSettings();
      this.isEnabled = result.isEnabled !== false;
      this.currentLevel = result.currentLevel || 'A';
      this.userProfile = result.userProfile || {};
    } catch (error) {
      console.error('Помилка завантаження налаштувань:', error);
      this.currentSettings = this.getDefaultSettings();
    }
  }

  getDefaultSettings() {
    return {
      // Рівень A
      'color-duplication': true,
      'alt-text': true,
      'animation-control': true,
      'large-targets': true,
      'simplified-view': false,

      // Рівень AA
      'auto-contrast': true,
      'text-zoom': true,
      'reflow-mode': true,
      'typography-settings': false,

      // Рівень AAA
      'ultra-contrast': false,
      'text-ocr': false,
      'context-adaptation': true,
```

```
        'cognitive-simplification': false,

        // Параметри
        'targetSize': 44,
        'contrastLevel': 2,
        'fontSize': 100,
        'lineHeight': 1.5,
        'letterSpacing': 0,
        'wordSpacing': 0,
        'fontFamily': 'system',
        'currentTheme': 'light'
    };
}

initAdapters() {
    try {
        // Ініціалізація адаптерів з перевіркою наявності класів
        if (typeof WCAGAdapter !== 'undefined') {
            this.adapters.set('wcag', new WCAGAdapter(this.currentSettings));
        } else {
            console.warn('WCAGAdapter не знайдено');
        }

        if (typeof ISOAdapter !== 'undefined') {
            this.adapters.set('iso', new ISOAdapter(this.currentSettings));
        } else {
            console.warn('ISOAdapter не знайдено');
        }

        if (typeof ContextDetector !== 'undefined') {
            this.adapters.set('context', new ContextDetector());
        } else {
            console.warn('ContextDetector не знайдено');
        }
    } catch (error) {
        console.error('Помилка ініціалізації адаптерів:', error);
    }
}

initUI() {
    try {
        // Ініціалізація UI елементів з перевіркою наявності класів
        if (typeof FloatingPanel !== 'undefined') {
            this.floatingPanel = new FloatingPanel(this);
        } else {
            console.warn('FloatingPanel не знайдено');
        }

        if (typeof ContentUI !== 'undefined') {
            this.contentUI = new ContentUI(this);
        } else {
            console.warn('ContentUI не знайдено');
        }
    } catch (error) {
        console.error('Помилка ініціалізації UI:', error);
    }
}

setupMessageListeners() {
    chrome.runtime.onMessage.addListener((request, sender, sendResponse) => {
        this.handleMessage(request, sender, sendResponse);
        return true; // Дозволяємо асинхронну відповідь
    });
}

async handleMessage(request, sender, sendResponse) {
    try {
        switch (request.action) {
            case 'updateSettings':
```

```
                await this.updateSettings(request.settings);
                sendResponse({ success: true, applied: true });
                break;

            case 'changeLevel':
                await this.changeLevel(request.level);
                sendResponse({ success: true });
                break;

            case 'changeContext':
                await this.changeContext(request.context);
                sendResponse({ success: true });
                break;

            case 'applyAIRecommendation':
                await this.applyAIRecommendation(request.recommendation);
                sendResponse({ success: true });
                break;

            case 'getState':
                const state = this.collectCurrentState();
                sendResponse(state);
                break;

            case 'toggleExtension':
                await this.toggleExtension();
                sendResponse({ success: true });
                break;

            case 'ping':
                sendResponse({ success: true, ready: true });
                break;

            case 'applyChanges':
                await this.applyAllAdaptations();
                await this.saveSettings();
                sendResponse({ success: true, applied: true });
                break;

            case 'getTrackerStatus':
                const trackerStatus = this.behaviorTracker ? this.behaviorTracker.getStatus() : { error: 'Tracker not initialized' };
                sendResponse({ success: true, trackerStatus });
                break;

            default:
                sendResponse({ success: false, error: 'Невідома дія' });
        }
    } catch (error) {
        console.error('Accessibility Adaptor: Помилка обробки повідомлення:', error);
        sendResponse({
            success: false,
            error: error.message,
            type: error.name
        });
    }
}

async updateSettings(newSettings) {
    this.currentSettings = { ...this.currentSettings, ...newSettings };
    this.pendingChanges.add('settings');

    await this.applyAllAdaptations();
    await this.saveSettings();

    this.pendingChanges.delete('settings');
}

async changeLevel(level) {
    this.currentLevel = level;
```

```
    await chrome.storage.local.set({ currentLevel: level });
    await this.applyAllAdaptations();
}

async changeContext(context) {
    // Логіка зміни контексту
    console.log('Зміна контексту на:', context);
    await chrome.storage.local.set({ currentContext: context });
}

async applyAllAdaptations() {
    if (!this.isEnabled) return;

    console.log('Застосування адаптацій...', this.currentSettings);

    try {
        // Застосовуємо всі адаптації через відповідні адаптери
        this.adapters.forEach((adapter, name) => {
            try {
                if (typeof adapter.apply === 'function') {
                    adapter.apply(this.currentSettings);
                }
            } catch (error) {
                console.error(`Помилка в адаптері ${name}:`, error);
            }
        });

        // Застосовуємо CSS змінні для глобальних налаштувань
        this.applyGlobalStyles();

        // Оновлюємо UI
        if (this.floatingPanel && typeof this.floatingPanel.updateUI === 'function') {
            this.floatingPanel.updateUI();
        }

        console.log('Адаптації успішно застосовано');

    } catch (error) {
        console.error('Помилка застосування адаптацій:', error);
    }
}

applyGlobalStyles() {
    const root = document.documentElement;

    // Розмір тексту
    if (this.currentSettings.fontSize) {
        root.style.setProperty('--accessibility-font-size', `${this.currentSettings.fontSize}%`);
        document.body.style.fontSize = `${this.currentSettings.fontSize}%`;
    }

    // Міжрядковий інтервал
    if (this.currentSettings.lineHeight) {
        root.style.setProperty('--accessibility-line-height', this.currentSettings.lineHeight);
        document.body.style.lineHeight = this.currentSettings.lineHeight;
    }

    // Відстань між літерами
    if (this.currentSettings.letterSpacing) {
        root.style.setProperty('--accessibility-letter-spacing', `${this.currentSettings.letterSpacing}em`);
        document.body.style.letterSpacing = `${this.currentSettings.letterSpacing}em`;
    }

    // Контраст
    if (this.currentSettings.contrastLevel) {
        root.setAttribute('data-contrast-level', this.currentSettings.contrastLevel);

        // Застосовуємо фільтр контрасту
        const contrastValues = { 1: 1.2, 2: 1.5, 3: 2 };
```

```
        document.body.style.filter = `contrast(${contrastValues[this.currentSettings.contrastLevel] || 1})`;
    }
}

async saveSettings() {
    try {
        await chrome.storage.local.set({
            userSettings: this.currentSettings
        });
    } catch (error) {
        console.error('Помилка збереження налаштувань:', error);
    }
}

startBehaviorTracking() {
    try {
        if (typeof BehaviorTracker !== 'undefined') {
            this.behaviorTracker = new BehaviorTracker();
            if (typeof this.behaviorTracker.startTracking === 'function') {
                this.behaviorTracker.startTracking();
            }
        } else {
            console.warn('BehaviorTracker не знайдено');
        }
    } catch (error) {
        console.error('Помилка ініціалізації трекера:', error);
    }
}

collectCurrentState() {
    return {
        enabled: this.isEnabled,
        level: this.currentLevel,
        settings: this.currentSettings,
        pendingChanges: Array.from(this.pendingChanges),
        pageInfo: {
            url: window.location.href,
            title: document.title,
            type: this.detectPageType()
        },
        adapters: Array.from(this.adapters.keys()),
        uiComponents: {
            floatingPanel: !!this.floatingPanel,
            contentUI: !!this.contentUI
        }
    };
}

detectPageType() {
    try {
        const url = window.location.href;
        const content = document.body?.innerText || '';

        if (/youtube|vimeo|netflix/i.test(url)) return 'video';
        if (/news|article|blog/i.test(url) || content.length > 1000) return 'news';
        if (/facebook|twitter|instagram/i.test(url)) return 'social';
        if (/shop|store|buy/i.test(url)) return 'ecommerce';
        if (document.forms.length > 0) return 'form';

        return 'generic';
    } catch (error) {
        return 'unknown';
    }
}

detectUserContext() {
    try {
        if (document.querySelector('video:not([paused])')) return 'watching_video';
        if (document.activeElement?.tagName === 'INPUT') return 'filling_form';
```

```
        if (this.getScrollDepth() > 50) return 'reading';
        if (document.querySelector('.carousel, .slider')) return 'viewing_carousel';
        return 'browsing';
    } catch (error) {
        return 'unknown';
    }
}

getScrollDepth() {
    try {
        const scrollHeight = document.documentElement.scrollHeight - window.innerHeight;
        const scrolled = window.scrollY;
        return scrollHeight > 0 ? (scrolled / scrollHeight) * 100 : 0;
    } catch (error) {
        return 0;
    }
}

async toggleExtension() {
    this.isEnabled = !this.isEnabled;
    await chrome.storage.local.set({ isEnabled: this.isEnabled });

    if (this.isEnabled) {
        this.applyAllAdaptations();
        if (this.floatingPanel && typeof this.floatingPanel.show === 'function') {
            this.floatingPanel.show();
        }
    } else {
        this.removeAllAdaptations();
        if (this.floatingPanel && typeof this.floatingPanel.hide === 'function') {
            this.floatingPanel.hide();
        }
    }
}

removeAllAdaptations() {
    // Видаляємо адаптації з адаптерів
    this.adapters.forEach((adapter, name) => {
        try {
            if (typeof adapter.remove === 'function') {
                adapter.remove();
            }
        } catch (error) {
            console.error(`Помилка видалення адаптацій в ${name}:`, error);
        }
    });

    // Видаляємо глобальні стилі
    const root = document.documentElement;
    root.style.removeProperty('--accessibility-font-size');
    root.style.removeProperty('--accessibility-line-height');
    root.style.removeProperty('--accessibility-letter-spacing');
    root.removeAttribute('data-contrast-level');

    document.body.style.removeProperty('font-size');
    document.body.style.removeProperty('line-height');
    document.body.style.removeProperty('letter-spacing');
    document.body.style.removeProperty('filter');
}

async applyAIRecommendation(recommendation) {
    if (!recommendation || !recommendation.settings) {
        console.error('Некоректна рекомендація AI');
        return;
    }

    this.currentSettings = { ...this.currentSettings, ...recommendation.settings };
    await this.updateSettings(this.currentSettings);
```

```
        if (this.contentUI && typeof this.contentUI.showFeedback === 'function') {
            this.contentUI.showFeedback(`Застосовано рекомендацію AI: ${recommendation.name}`);
        }
    }

    hasPendingChanges() {
        return this.pendingChanges.size > 0;
    }

    async forceApplyChanges() {
        console.log('Примусове застосування змін...');
        await this.applyAllAdaptations();
        await this.saveSettings();

        if (this.floatingPanel && typeof this.floatingPanel.showAppliedFeedback === 'function') {
            this.floatingPanel.showAppliedFeedback('Зміни успішно застосовано!');
        }
    }
}

// Глобальні CSS для адаптацій
const globalAdaptationStyles = `
    :root {
        --accessibility-font-size: 100%;
        --accessibility-line-height: 1.5;
        --accessibility-letter-spacing: 0em;
    }

    [data-contrast-level="1"] {
        filter: contrast(1.2) !important;
    }

    [data-contrast-level="2"] {
        filter: contrast(1.5) !important;
    }

    [data-contrast-level="3"] {
        filter: contrast(2) !important;
    }

    .simplified-view * {
        background-image: none !important;
        animation: none !important;
        transition: none !important;
    }

    .simplified-view .ad,
    .simplified-view [class*="banner"],
    .simplified-view [class*="advertisement"],
    .simplified-view [class*="popup"],
    .simplified-view [class*="modal"] {
        display: none !important;
    }

    .accessibility-large-targets button,
    .accessibility-large-targets a,
    .accessibility-large-targets [role="button"],
    .accessibility-large-targets input[type="submit"],
    .accessibility-large-targets input[type="button"] {
        min-width: 44px !important;
        min-height: 44px !important;
        padding: 12px 8px !important;
    }

    .accessibility-high-contrast {
        background: #000 !important;
        color: #fff !important;
        border-color: #fff !important;
    }
```

```
    .accessibility-high-contrast * {
        background: #000 !important;
        color: #fff !important;
        border-color: #fff !important;
    }
`;

// Додаємо глобальні стилі
try {
    const styleElement = document.createElement('style');
    styleElement.textContent = globalAdaptationStyles;
    styleElement.id = 'accessibility-adaptor-styles';
    document.head.appendChild(styleElement);
} catch (error) {
    console.error('Помилка додавання глобальних стилів:', error);
}

// Безпечна ініціалізація
function initializeContentScript() {
    try {
        if (document.readyState === 'loading') {
            document.addEventListener('DOMContentLoaded', () => {
                window.accessibilityAdaptor = new ContentScript();
            });
        } else {
            window.accessibilityAdaptor = new ContentScript();
        }
    } catch (error) {
        console.error('Помилка ініціалізації ContentScript:', error);
    }
}

// Запускаємо ініціалізацію
initializeContentScript();
```

Backend

App.py

```
from flask import Flask, request, jsonify
from flask_cors import CORS
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime, timedelta
import pandas as pd
import numpy as np
import json
import os
from threading import Thread
import time

from config import config
from models.user_model import db, UserSession, BehaviorData, UserSettings, ModelPrediction
from models.lstm_model import ModelManager
from utils.normalized import DataNormalizer
from utils.feature_engineer import FeatureEngineer

def create_app(config_name='default'):
    app = Flask(__name__)
    app.config.from_object(config[config_name])

    # Ініціалізація розширень
    db.init_app(app)
    CORS(app, origins=app.config['CORS_ORIGINS'])

    # Ініціалізація менеджера моделей
    model_manager = ModelManager(app.config)

    with app.app_context():
        # Створення таблиць
        db.create_all()
```

```
        # Ініціалізація моделей
        try:
            model_manager.initialize_models()
            model_manager.load_models()
        except Exception as e:
            print(f"Помилка ініціалізації моделей: {e}")

    # Глобальна змінна для менеджера моделей
    app.model_manager = model_manager
    # Фонова задача для оновлення моделі
    def background_model_updater():
        with app.app_context():
            while True:
                try:
                    update_model(app)
                except Exception as e:
                    print(f"Помилка оновлення моделі: {e}")
                time.sleep(app.config['MODEL_UPDATE_INTERVAL'])

    # Запуск фонової задачі
    if not app.config['TESTING']:
        Thread(target=background_model_updater, daemon=True).start()

    return app

def update_model(app):
    print("Оновлення моделі...")
    """Оновлення моделі на нових даних"""
    try:
        # Отримуємо дані за останній тиждень
        week_ago = datetime.utcnow() - timedelta(days=1)

        behavior_data = BehaviorData.query.filter(
            BehaviorData.timestamp <= week_ago
        ).all()

        # if len(behavior_data) < 100:  # Мінімальна кількість даних
        #     print("Недостатньо даних для оновлення моделі")
        #     return

        # Конвертуємо в DataFrame
        data_dicts = [data.to_dict() for data in behavior_data]

        df = pd.DataFrame(data_dicts)
        columns_to_drop = ['additional_data', 'page_type_preferences']
        df = df.drop(columns=columns_to_drop, errors='ignore')
        # Оновлюємо модель
        app.model_manager.train_complete_pipeline(df)
        print("Модель успішно оновлено")

    except Exception as e:
        print(f"Помилка оновлення моделі: {e}")

# API Routes
app = create_app()

@app.route('/api/health', methods=['GET'])
def health_check():
    """Перевірка стану сервера"""
    return jsonify({
        'status': 'healthy',
        'timestamp': datetime.utcnow().isoformat(),
        'model_loaded': app.model_manager.is_trained
    })

@app.route('/api/behavior', methods=['POST'])
def receive_behavior_data():
    """Отримання даних поведінки від розширення"""
```

```
    try:
        print("Отримання даних поведінки...")
        data = request.get_json()
        print(data)

        if not data:
            return jsonify({'error': 'No data provided'}), 400

        # Перевірка обов'язкових полів
        required_fields = ['sessionId', 'data', 'timestamp']
        for field in required_fields:
            if field not in data:
                return jsonify({'error': f'Missing required field: {field}'}), 400

        # Отримання або створення сесії
        session = UserSession.query.get(data['sessionId'])
        if not session:
            session = UserSession(
                id=data['sessionId'],
                user_id=data.get('userId'),
                user_agent=request.headers.get('User-Agent'),
                ip_address=request.remote_addr
            )
            db.session.add(session)

        # Збереження даних поведінки
        for behavior_point in data['data']:
            behavior_data = BehaviorData(
                session_id=session.id,
                timestamp=datetime.fromisoformat(behavior_point['timestamp'].replace('Z', '+00:00')),
                event_type=behavior_point.get('type', 'state_capture'),
                page_type=behavior_point.get('pageType', 'generic'),
                user_context=behavior_point.get('userContext', 'browsing'),
                font_size=behavior_point.get('fontSize'),
                contrast_level=behavior_point.get('contrastLevel'),
                zoom_level=behavior_point.get('zoomLevel'),
                theme=behavior_point.get('theme'),
                line_height=behavior_point.get('lineHeight'),
                letter_spacing=behavior_point.get('letterSpacing'),
                animation_state=behavior_point.get('animationState', True),
                simplified_view=behavior_point.get('simplifiedView', False),
                scroll_depth=behavior_point.get('scrollDepth', 0),
                scroll_speed=behavior_point.get('scrollSpeed', 0),
                click_count=behavior_point.get('clicks', 0),
                key_stroke_count=behavior_point.get('keyStrokes', 0),
                time_on_page=behavior_point.get('timeOnPage', 0),
                viewport_width=behavior_point.get('viewport', {}).get('width'),
                viewport_height=behavior_point.get('viewport', {}).get('height'),
                url=behavior_point.get('url'),
                focus_element=behavior_point.get('focusElement'),
                additional_data=json.dumps(behavior_point.get('additionalData', {}))
            )
            db.session.add(behavior_data)

        db.session.commit()

        return jsonify({'success': True, 'records_saved': len(data['data'])})

    except Exception as e:
        db.session.rollback()
        return jsonify({'error': str(e)}), 500

@app.route('/api/settings', methods=['POST'])
def receive_user_settings():
    """Отримання налаштувань користувача"""
    try:
        data = request.get_json()

        if not data:
```

```
            return jsonify({'error': 'No data provided'}), 400

        # Збереження налаштувань
        settings = UserSettings(
            session_id=data.get('sessionId'),
            user_id=data.get('userId'),
            color_duplication=data.get('colorDuplication', True),
            alt_text=data.get('altText', True),
            animation_control=data.get('animationControl', True),
            large_targets=data.get('largeTargets', True),
            simplified_view=data.get('simplifiedView', False),
            auto_contrast=data.get('autoContrast', True),
            text_zoom=data.get('textZoom', True),
            reflow_mode=data.get('reflowMode', True),
            ultra_contrast=data.get('ultraContrast', False),
            text_ocr=data.get('textOcr', False),
            context_adaptation=data.get('contextAdaptation', True),
            cognitive_simplification=data.get('cognitiveSimplification', False),
            target_size=data.get('targetSize', 44),
            contrast_level=data.get('contrastLevel', 2),
            font_size=data.get('fontSize', 100.0),
            line_height=data.get('lineHeight', 1.5),
            letter_spacing=data.get('letterSpacing', 0.0),
            word_spacing=data.get('wordSpacing', 0.0),
            font_family=data.get('fontFamily', 'system'),
            current_theme=data.get('currentTheme', 'light'),
            context_mode=data.get('contextMode', 'auto'),
            page_type_preferences=json.dumps(data.get('pageTypePreferences', {}))
        )

        db.session.add(settings)
        db.session.commit()

        return jsonify({'success': True})

    except Exception as e:
        db.session.rollback()
        return jsonify({'error': str(e)}), 500

@app.route('/api/recommendations', methods=['GET'])
def get_recommendations():
    """Отримання рекомендацій AI для користувача"""
    try:
        user_id = request.args.get('userId')
        session_id = request.args.get('sessionId')
        page_type = request.args.get('pageType', 'generic')
        user_context = request.args.get('userContext', 'browsing')

        if not user_id and not session_id:
            return jsonify({'error': 'Either userId or sessionId is required'}), 400

        # Отримання історії поведінки користувача
        query = BehaviorData.query

        if user_id:
            # Знаходимо сесії користувача
            user_sessions = UserSession.query.filter_by(user_id=user_id).all()
            session_ids = [session.id for session in user_sessions]
            query = query.filter(BehaviorData.session_id.in_(session_ids))
        else:
            query = query.filter_by(session_id=session_id)

        # Останні 50 записів
        behavior_history = query.order_by(BehaviorData.timestamp.desc()).limit(50).all()

        if not behavior_history:
            # Повертаємо базові рекомендації, якщо немає даних
            return jsonify({
                'success': True,
```

```
            'recommendation': {
                'name': 'Базові налаштування',
                'settings': {
                    'fontSize': 100,
                    'contrastLevel': 2,
                    'simplifiedView': False,
                    'animationControl': True,
                    'largeTargets': True
                },
                'confidence': 0.5,
                'isFallback': True
            }
        })

        # Конвертуємо в DataFrame
        behavior_dicts = [data.to_dict() for data in behavior_history]
        df = pd.DataFrame(behavior_dicts)

        # Отримуємо прогноз
        if app.model_manager.is_trained:
            prediction = app.model_manager.predict_settings(df, page_type, user_context)

            # Зберігаємо прогноз
            model_prediction = ModelPrediction(
                user_id=user_id,
                page_type=page_type,
                user_context=user_context,
                predicted_settings=json.dumps(prediction['predicted_settings']),
                confidence_scores=json.dumps(prediction['confidence_scores']),
                model_version='1.0'
            )
            db.session.add(model_prediction)
            db.session.commit()

            return jsonify({
                'success': True,
                'recommendation': {
                    'name': 'Персоналізовані налаштування',
                    'settings': prediction['predicted_settings'],
                    'confidence_scores': prediction['confidence_scores'],
                    'raw_prediction': prediction.get('raw_prediction', []),
                    'isFallback': False
                }
            })
        else:
            # Модель не навчена - повертаємо статистичні рекомендації
            return get_statistical_recommendations(df, page_type, user_context)

    except Exception as e:
        db.session.rollback()
        return jsonify({'error': str(e)}), 500

def get_statistical_recommendations(df, page_type, user_context):
    """Статистичні рекомендації, коли модель не навчена"""
    # Аналізуємо історію користувача
    avg_font_size = df['font_size'].mean()
    common_contrast = df['contrast_level'].mode()[0] if not df['contrast_level'].mode().empty else 2

    # Адаптація під контекст
    context_settings = {
        'reading': {'fontSize': max(125, avg_font_size), 'simplifiedView': True},
        'filling_form': {'largeTargets': True, 'contrastLevel': max(common_contrast, 2)},
        'watching_video': {'animationControl': False, 'simplifiedView': True},
        'browsing': {'fontSize': avg_font_size, 'contrastLevel': common_contrast}
    }

    base_settings = {
        'fontSize': avg_font_size,
        'contrastLevel': common_contrast,
```

```
        'simplifiedView': False,
        'animationControl': True,
        'largeTargets': True
    }

    # Застосовуємо контекстні налаштування
    context_adjustment = context_settings.get(user_context, {})
    recommended_settings = {**base_settings, **context_adjustment}

    return jsonify({
        'success': True,
        'recommendation': {
            'name': 'Статистичні рекомендації',
            'settings': recommended_settings,
            'confidence': 0.7,
            'isFallback': True
        }
    })

@app.route('/api/feedback', methods=['POST'])
def receive_feedback():
    """Отримання зворотного зв'язку від користувача"""
    try:
        data = request.get_json()

        prediction_id = data.get('predictionId')
        feedback = data.get('feedback')  # 'positive', 'negative', 'neutral'
        applied = data.get('applied', False)

        if not prediction_id:
            return jsonify({'error': 'predictionId is required'}), 400

        # Оновлюємо прогноз
        prediction = ModelPrediction.query.get(prediction_id)
        if prediction:
            prediction.applied = applied
            prediction.user_feedback = feedback
            db.session.commit()

        return jsonify({'success': True})

    except Exception as e:
        db.session.rollback()
        return jsonify({'error': str(e)}), 500

@app.route('/api/model/status', methods=['GET'])
def get_model_status():
    """Статус моделі ML"""
    return jsonify({
        'is_trained': app.model_manager.is_trained,
        'last_training': getattr(app.model_manager, 'last_training_time', None),
        'model_metrics': getattr(app.model_manager.model, 'last_metrics', {})
    })

@app.route('/api/data/summary', methods=['GET'])
def get_data_summary():
    """Статистика даних"""
    total_sessions = UserSession.query.count()
    total_behavior_records = BehaviorData.query.count()
    total_settings_records = UserSettings.query.count()
    total_predictions = ModelPrediction.query.count()

    return jsonify({
        'sessions': total_sessions,
        'behavior_records': total_behavior_records,
        'settings_records': total_settings_records,
        'predictions': total_predictions
    })
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=app.config['DEBUG'])
```

ltm_model.py

```
from flask import json
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout, BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from keras.regularizers import l2
import numpy as np
import pandas as pd
import joblib
import os
from datetime import datetime
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

class AccessibilityLSTMModel:
    def __init__(self, config):
        self.config = config
        self.model = None
        self.sequence_length = config.get('SEQUENCE_LENGTH', 10)
        self.input_dim = None
        self.output_dim = None

    def build_model(self, input_dim, output_dim):
        """Побудова архітектури LSTM моделі"""
        self.input_dim = input_dim
        self.output_dim = output_dim

        self.model = Sequential([
            # Перший LSTM шар
            LSTM(128,
                 return_sequences=True,
                 input_shape=(self.sequence_length, input_dim),
                 kernel_regularizer=l2(0.001),
                 recurrent_regularizer=l2(0.001)),
            BatchNormalization(),
            Dropout(0.3),

            # Другий LSTM шар
            LSTM(64,
                 return_sequences=False,
                 kernel_regularizer=l2(0.001),
                 recurrent_regularizer=l2(0.001)),
            BatchNormalization(),
            Dropout(0.3),

            # Повнозв'язні шари
            Dense(32, activation='relu', kernel_regularizer=l2(0.001)),
            BatchNormalization(),
            Dropout(0.2),

            Dense(16, activation='relu', kernel_regularizer=l2(0.001)),
            Dropout(0.1),

            # Вихідний шар
            Dense(output_dim, activation='linear')
        ])

        # Компіляція моделі
        self.model.compile(
            optimizer=Adam(learning_rate=0.001),
            loss='mse',
            metrics=['mae', 'mse']
```

```
        )

        return self.model

    def train(self, X_train, y_train, X_val=None, y_val=None):
        """Навчання моделі"""
        if self.model is None:
            self.build_model(X_train.shape[2], y_train.shape[1])

        # Callbacks
        callbacks = [
            EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True),
            ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10, min_lr=1e-7),
            ModelCheckpoint(
                os.path.join(self.config['MODELS_PATH'], 'best_lstm_model.keras'),
                monitor='val_loss',
                save_best_only=True,
                save_weights_only=False
            )
        ]

        validation_data = (X_val, y_val) if X_val is not None else None

        # Навчання
        history = self.model.fit(
            X_train, y_train,
            batch_size=self.config['BATCH_SIZE'],
            epochs=self.config['EPOCHS'],
            validation_data=validation_data,
            callbacks=callbacks,
            verbose=1,
            shuffle=True
        )

        return history

    def predict(self, X):
        """Прогнозування налаштувань"""
        if self.model is None:
            raise ValueError("Модель не навчена. Спочатку викличте train() або load_model()")

        predictions = self.model.predict(X)
        return predictions

    def predict_for_user(self, user_sequence, normalizer, feature_engineer):
        """Прогнозування для конкретного користувача"""
        # Підготовка даних
        processed_data = feature_engineer.engineer_features(user_sequence)
        normalized_data = normalizer.transform(processed_data)
        print("\n=== Нормалізовані нові дані ===")
        print(normalized_data.head(20))
        # Створення послідовності
        if len(normalized_data) < self.sequence_length:
            # Якщо недостатньо даних, використовуємо padding
            padding = np.zeros((self.sequence_length - len(normalized_data), normalized_data.shape[1]))
            sequence = np.vstack([padding, normalized_data])
        else:
            sequence = normalized_data[-self.sequence_length:]

        sequence = sequence.reshape(1, self.sequence_length, -1)

        # Прогнозування
        prediction = self.predict(sequence)[0]

        # Інтерпретація прогнозу
        interpreted_prediction = self._interpret_prediction(prediction)

        return interpreted_prediction
```

```
def _interpret_prediction(self, prediction):
    """Інтерпретація вихідних даних моделі"""
    # prediction: [font_size, contrast_level, simplified_view, animation_control, large_targets]

    # Денормалізація числових значень
    font_size = max(100, min(400, prediction[0] * 50 + 100))  # 100-400%
    contrast_level = max(1, min(3, round(prediction[1] * 2 + 1)))  # 1-3

    # Бінарні налаштування з порогом
    simplified_view = prediction[2] > 0.5
    animation_control = prediction[3] > 0.5
    # large_targets = prediction[4] > 0.5

    # Рівні довіри
    confidence_scores = {
        'font_size': min(1.0, abs(prediction[0]) / 2),
        'contrast_level': min(1.0, abs(prediction[1])),
        'simplified_view': abs(prediction[2] - 0.5) * 2,
        'animation_control': abs(prediction[3] - 0.5) * 2,
        # 'large_targets': abs(prediction[4] - 0.5) * 2
    }

    return {
        'predicted_settings': {
            'fontSize': font_size,
            'contrastLevel': contrast_level,
            'simplifiedView': simplified_view,
            'animationControl': animation_control,
            # 'largeTargets': large_targets
        },
        'confidence_scores': confidence_scores,
        'raw_prediction': prediction.tolist()
    }

def save_model(self, filepath):
    """Збереження моделі"""
    if self.model is None:
        raise ValueError("Модель не ініціалізована")

    os.makedirs(os.path.dirname(filepath), exist_ok=True)
    self.model.save(filepath)

    # Зберігаємо метадані
    metadata = {
        'input_dim': self.input_dim,
        'output_dim': self.output_dim,
        'sequence_length': self.sequence_length,
        'timestamp': datetime.now().isoformat()
    }

    joblib.dump(metadata, filepath.replace('.keras', '_metadata.pkl'))

def load_model(self, filepath):
    """Завантаження моделі"""
    if not os.path.exists(filepath):
        raise FileNotFoundError(f"Файл моделі не знайдено: {filepath}")

    # Завантажуємо модель
    self.model = tf.keras.models.load_model(filepath)

    # Завантажуємо метадані
    metadata_path = filepath.replace('.keras', '_metadata.pkl')
    if os.path.exists(metadata_path):
        metadata = joblib.load(metadata_path)
        self.input_dim = metadata['input_dim']
        self.output_dim = metadata['output_dim']
        self.sequence_length = metadata['sequence_length']

    return self.model
```

```
    def evaluate(self, X_test, y_test):
        """Оцінка моделі"""
        if self.model is None:
            raise ValueError("Модель не навчена")

        evaluation = self.model.evaluate(X_test, y_test, verbose=0)
        return dict(zip(self.model.metrics_names, evaluation))

class ModelManager:
    def __init__(self, config):
        self.config = config
        self.model = None
        self.normalizer = None
        self.feature_engineer = None
        self.is_trained = False

    def initialize_models(self):
        """Ініціалізація всіх моделей та процесорів"""
        from utils.normalized import DataNormalizer, SequenceNormalizer
        from utils.feature_engineer import FeatureEngineer

        self.normalizer = DataNormalizer(self.config)
        self.feature_engineer = FeatureEngineer()
        self.sequence_normalizer = SequenceNormalizer(self.config['SEQUENCE_LENGTH'])
        self.model = AccessibilityLSTMModel(self.config)

    def train_complete_pipeline(self, df):
        """Повний пайплайн навчання"""
        # Інженерія features
        df_engineered = self.feature_engineer.engineer_features(df)

        # Нормалізація
        df_normalized = self.normalizer.fit(df_engineered)
        # Підготовка даних для LSTM
        sequences = []
        targets = []
        for user_id in df_normalized['user_id'].unique():
            user_sequences, user_targets = self.sequence_normalizer.create_sequences(
                df_normalized, user_id
            )

            sequences.extend(user_sequences)
            targets.extend(user_targets)

        if len(sequences) == 0:
            raise ValueError("Недостатньо даних для навчання")

        X = np.array(sequences)
        y = np.array(targets)

        # Розділення на train/validation
        split_idx = int(0.8 * len(X))
        X_train, X_val = X[:split_idx], X[split_idx:]
        y_train, y_val = y[:split_idx], y[split_idx:]

        # Навчання моделі
        self.model.build_model(X_train.shape[2], y_train.shape[1])
        history = self.model.train(X_train, y_train, X_val, y_val)

        self.evaluate_model(X_val, y_val, history)
        self.plot_training(history)
        y_pred = self.model.model.predict(X_val)

        self.plot_per_feature_mae(y_val, y_pred)
        self.plot_real_vs_predicted(y_val, y_pred, 0, 'font_size')
        self.plot_real_vs_predicted(y_val, y_pred, 1, 'contrast_level')
        self.plot_real_vs_predicted(y_val, y_pred, 2, 'simplified_view')
        self.plot_real_vs_predicted(y_val, y_pred, 3, 'animation_control')
```

```
        self.is_trained = True

        # Збереження моделей
        self.save_models()

        return history

    def evaluate_model(self, X_val, y_val, history):
        print("=== Оцінка ===")
        val_loss, val_mae, val_mse = self.model.model.evaluate(X_val, y_val)

        print("=== Підсумкова оцінка моделі на валідації ===")
        print(f"val_loss (MSE): {val_loss:.4f}")
        print(f"val_MAE:        {val_mae:.4f}")
        print(f"val_MSE:        {val_mse:.4f}")

        print("=== Приклади прогнозів ===")
        y_pred = self.model.model.predict(X_val[:10])
        print("Real:", y_val[:10])
        print("Pred:", y_pred[:10])

        loss = history.history['loss']
        val_loss = history.history['val_loss']

        mae = history.history['mae']
        val_mae = history.history['val_mae']

        return val_loss, val_mae, val_mse
    def plot_training(self, history, save_dir='data/plots'):
        os.makedirs(save_dir, exist_ok=True)

        loss = history.history['loss']
        val_loss = history.history['val_loss']
        mae = history.history['mae']
        val_mae = history.history['val_mae']

        # --- LOSS ---
        plt.figure(figsize=(8, 4))
        plt.plot(loss, label='Train Loss')
        plt.plot(val_loss, label='Val Loss')
        plt.title('Динаміка MSE під час навчання')
        plt.xlabel('Епоха')
        plt.ylabel('MSE')
        plt.legend()
        plt.grid(True)
        plt.tight_layout()
        plt.savefig(os.path.join(save_dir, 'loss_curve.png'))  # ✅ зберігаємо
        plt.close()

        # --- MAE ---
        plt.figure(figsize=(8, 4))
        plt.plot(mae, label='Train MAE')
        plt.plot(val_mae, label='Val MAE')
        plt.title('Середня абсолютна помилка')
        plt.xlabel('Епоха')
        plt.ylabel('MAE')
        plt.legend()
        plt.grid(True)
        plt.tight_layout()
        plt.savefig(os.path.join(save_dir, 'mae_curve.png'))  # ✅ зберігаємо
        plt.close()

    def plot_real_vs_predicted(self, y_true, y_pred, idx, name, save_dir='data/plots'):

        os.makedirs(save_dir, exist_ok=True)

        plt.figure(figsize=(5, 5))
        plt.scatter(y_true[:, idx], y_pred[:, idx], s=10)
```

```
        min_val = min(y_true[:, idx].min(), y_pred[:, idx].min())
        max_val = max(y_true[:, idx].max(), y_pred[:, idx].max())
        plt.plot([min_val, max_val], [min_val, max_val])

        plt.xlabel('Реальні значення')
        plt.ylabel('Прогнозовані значення')
        plt.title(f'Порівняння істинних і прогнозованих значень: {name}')
        plt.tight_layout()

        # збережемо окремий файл під кожну ознаку
        plt.savefig(os.path.join(save_dir, f'real_predict_{name}.png'))
        plt.close()

    def plot_per_feature_mae(self, y_true, y_pred, save_path='data/plots/per_feature_mae.png'):
        all_feature_names = ["font_size", "contrast_level",
                             "simplified_view", "animation_control", "large_targets"]

        n_outputs = y_true.shape[1]
        feature_names = all_feature_names[:n_outputs]  # підрізаємо до 4, якщо потрібно

        mae_per_feature = np.mean(np.abs(y_true - y_pred), axis=0)

        plt.figure(figsize=(7, 4))
        plt.bar(range(len(feature_names)), mae_per_feature)
        plt.xticks(range(len(feature_names)), feature_names, rotation=30, ha='right')
        plt.ylabel('MAE')
        plt.title('MAE для кожного прогнозованого параметра')
        plt.tight_layout()

        os.makedirs(os.path.dirname(save_path), exist_ok=True)
        plt.savefig(save_path)
        plt.close()

    def predict_settings(self, user_data, page_type, user_context):
        """Прогнозування налаштувань для користувача"""
        if not self.is_trained:
            raise ValueError("Модель не навчена")

        # Додаємо контекстні дані
        user_data = user_data.copy()
        user_data['page_type'] = page_type
        user_data['user_context'] = user_context

        # Прогнозування
        prediction = self.model.predict_for_user(
            user_data, self.normalizer, self.feature_engineer
        )

        return prediction

    def save_models(self):
        """Збереження всіх моделей"""
        model_path = os.path.join(self.config['MODELS_PATH'], 'lstm_model.keras')
        normalizer_path = os.path.join(self.config['MODELS_PATH'], 'normalizer.pkl')

        self.model.save_model(model_path)
        self.normalizer.save(normalizer_path)

        print(f"Моделі збережено: {model_path}, {normalizer_path}")

    def load_models(self):
```

normalized.py

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.impute import SimpleImputer
import joblib
import json
```

```
import os

class DataNormalizer:
    def __init__(self, config):
        self.config = config
        self.scalers = {}
        self.encoders = {}
        self.imputers = {}
        self.feature_names = []

    def fit(self, df):
        """Навчання нормалізаторів на тренувальних даних"""
        # Визначаємо типи features
        numerical_features = self._get_numerical_features(df)
        categorical_features = self._get_categorical_features(df)
        boolean_features = self._get_boolean_features(df)

        # Нормалізація числових features
        for feature in numerical_features:
            scaler = StandardScaler()
            df[feature] = scaler.fit_transform(df[[feature]])
            self.scalers[feature] = scaler

        # Кодування категоріальних features
        for feature in categorical_features:
            encoder = LabelEncoder()
            df[feature] = encoder.fit_transform(df[feature].astype(str))
            self.encoders[feature] = encoder

        # Обробка булевих features
        for feature in boolean_features:
            df[feature] = df[feature].astype(int)

        self.feature_names = list(df.columns)
        return df

    def transform(self, df):
        """Трансформація нових даних"""
        # Numerical features
        for feature, scaler in self.scalers.items():
            if feature in df.columns:
                df[feature] = scaler.transform(df[[feature]])

        # Categorical features
        for feature, encoder in self.encoders.items():
            if feature in df.columns:
                # Обробка нових категорій
                df[feature] = df[feature].astype(str)
                unknown_categories = set(df[feature]) - set(encoder.classes_)
                if unknown_categories:
                    # Замінюємо невідомі категорії на найпоширенішу
                    df[feature] = df[feature].apply(
                        lambda x: encoder.transform([x])[0] if x in encoder.classes_ else 0
                    )
                else:
                    df[feature] = encoder.transform(df[feature])

        # Boolean features
        boolean_features = self._get_boolean_features(df)
        for feature in boolean_features:
            df[feature] = df[feature].astype(int)

        return df

    def _get_numerical_features(self, df):
        numerical = []
        for col in df.columns:
            if pd.api.types.is_numeric_dtype(df[col]) and df[col].nunique() > 2:
                numerical.append(col)
```

```
        return numerical

    def _get_categorical_features(self, df):
        categorical = []
        for col in df.columns:
            if pd.api.types.is_object_dtype(df[col]) or df[col].nunique() <= 10:
                categorical.append(col)
        return categorical

    def _get_boolean_features(self, df):
        boolean = []
        for col in df.columns:
            if df[col].nunique() == 2 and set(df[col].unique()) <= {0, 1, True, False}:
                boolean.append(col)
        return boolean

    def save(self, filepath):
        """Збереження нормалізаторів"""
        os.makedirs(os.path.dirname(filepath), exist_ok=True)
        joblib.dump({
            'scalers': self.scalers,
            'encoders': self.encoders,
            'imputers': self.imputers,
            'feature_names': self.feature_names
        }, filepath)

    def load(self, filepath):
        """Завантаження нормалізаторів"""
        data = joblib.load(filepath)
        self.scalers = data['scalers']
        self.encoders = data['encoders']
        self.imputers = data['imputers']
        self.feature_names = data['feature_names']

class SequenceNormalizer:
    def __init__(self, sequence_length=10):
        self.sequence_length = sequence_length
        self.feature_normalizer = DataNormalizer({})

    def create_sequences(self, df, user_id):
        """Створення послідовностей для LSTM"""
        sequences = []
        targets = []

        # Групуємо за користувачем
        user_data = df[df['user_id'] == user_id].sort_values('timestamp')
        if len(user_data) < self.sequence_length:
            print(f"Недостатньо даних для користувача {user_id} для створення послідовностей.")
            return sequences, targets

        # Створюємо послідовності
        for i in range(len(user_data) - self.sequence_length):
            sequence = user_data.iloc[i:i + self.sequence_length]
            target = user_data.iloc[i + self.sequence_length]

            # Features для послідовності (поведінка + контекст)
            sequence_features = self._extract_sequence_features(sequence)

            # Target (налаштування)
            target_features = self._extract_target_features(target)

            sequences.append(sequence_features)
            targets.append(target_features)

        return np.array(sequences), np.array(targets)

    def _extract_sequence_features(self, sequence):
        """
```

Видобуток features з послідовності для LSTM.Повертаємо масив форми (sequence_length, n_features),тобто окремий вектор ознак для кожного кроку часу.

```
        """
        # Базові числові ознаки для кожного кроку
        base = sequence[[
              'scroll_depth',
              'scroll_speed',
              'click_count',
              'time_on_page',
              'font_size',
              'contrast_level'
           ]].copy()

        # Закодуємо контекст по кожному рядку
        base['page_type_enc'] = sequence['page_type'].map(self._encode_page_type)
        base['user_context_enc'] = sequence['user_context'].map(self._encode_user_context)
        return base.to_numpy()


    def _extract_target_features(self, target):
        """Видобуток target features"""
        return np.array([
              float(target['font_size']),
              float(target['contrast_level']),
              int(target['simplified_view']),
              int(target['animation_state']),
              # якщо є стовпець large_targets у df – можна додати:
              # int(target['large_targets'])
           ], dtype=float)

    def _encode_page_type(self, page_type):
        encoding = {
           'news': 0, 'social': 1, 'ecommerce': 2,
           'form': 3, 'video': 4, 'generic': 5
        }
        return encoding.get(page_type, 5)

    def _encode_user_context(self, user_context):
        encoding = {
           'reading': 0, 'filling_form': 1, 'watching_video': 2,
           'browsing': 3, 'viewing_carousel': 4
        }
        return encoding.get(user_context, 3)
```