Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Допущено до захисту»
Завідувач кафедри
комп'ютерних наук
доктор технічних наук, професор
(науковий ступінь, наукове звання)
Бондарчук А. П.
(прізвище, ініціали) (підпис)
« ___ » _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «Магістр»
Спеціальність 122 Комп'ютерні науки
Освітня програма 122.00.02 Інформаційно-аналітичні системи

Тема роботи Розробка персоналізованої рекомендаційної системи для підбору страв у мобільному застосунку з використанням методів машинного навчання

Виконав
студент групи ІАСм-1-24-1.4д
(шифр академічної групи)
Карлов Євгеній Олександрович
(прізвище, ім'я, по батькові)
(підпис)

Науковий керівник
кандидат технічних наук, доцент
(науковий ступінь, наукове звання)
Яскевич В. О.
(прізвище, ініціали)
(підпис)

Київ – 2025

Київський столичний університет імені Бориса Грінченка

Факультет інформаційних технологій та математики

Кафедра комп'ютерних наук

«Затверджую»
Завідувач кафедри
комп'ютерних наук
кандидат технічних наук, доцент
(науковий ступінь, наукове звання)
Машкіна І. В.
(прізвище, ініціали) (підпис)
« ___ » _____ 2024 р.

ЗАВДАННЯ НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

студенту групи ІАСм-1-24-1.4д

Карлову Євгенію Олександровичу
(прізвище, ім'я, по батькові)

Тема роботи Розробка персоналізованої рекомендаційної системи для підбору страв у мобільному застосунку з використанням методів машинного навчання

1. Вихідні дані: мова програмування Python, мова програмування Kotlin, бібліотека TensorFlow, бібліотека TensorFlow Lite, фреймворк Kotlin Mutliplatform.

2. Основні завдання: провести аналіз методів побудови рекомендаційних систем; визначити фактори, що впливають на вибір страв, провести збір та обробку даних; розробити алгоритми роботи рекомендаційної системи та створити модель машинного навчання; розробити мобільний застосунок з виведенням висновків на пристрої; протестувати різні конфігурації моделі, інтегрувати модель та протестувати прототип.

3. Пояснювальна записка: обсяг до 65 сторінок формату А4, комп'ютерного набору, з дотриманням вимог стандарту і методичних рекомендацій кафедри.

4. Графічні матеріали: презентація.

5. Додатки: зображення структурних схем системи, запис роботи прототипу, вихідний код.

6. Строк подання роботи на кафедру: «01» 12. 2025 р

РЕФЕРАТ

Дипломна робота: 71 с, 2 рис, 2 табл., 82 посилань.

Актуальність: робота присвячена вирішенню проблеми підвищення якості та безпеки персоналізованих рекомендацій.

Об’єкт дослідження: персоналізовані рекомендаційні системи на користувацьких пристроях.

Предмет дослідження: методи побудови компактних векторних представлень мультимодальних даних страв та алгоритми їх ефективної обробки на мобільних пристроях.

Мета роботи: підвищити ефективність та приватність рекомендаційних систем шляхом розробки методу виведення висновків на пристрої користувача.

Завдання роботи:

- провести аналіз існуючих підходів та методів машинного навчання для побудови рекомендаційних систем;
- визначити ключові фактори, що впливають на персоналізований вибір страв, провести збір та обробку даних;
- розробити алгоритми роботи рекомендаційної системи та створити модель машинного навчання;
- розробити мультиплатформенний мобільний застосунок з виведенням висновків на пристрої;
- протестувати конфігурації моделі, інтегрувати модель та протестувати прототип.

Методи дослідження: аналіз наукової літератури, порівняльний аналіз, статистичне опрацювання даних, математичне моделювання, експериментальна оцінка.

Наукова новизна дослідження полягає в удосконаленні підходів до застосування рекомендаційних систем на користувацьких пристроях.

Практичне значення дослідження: створено прототип програмного продукту, який може бути адаптований та впроваджений в харчові сервіси.

Ключові слова: рекомендаційна система, інгредієнт, персоналізація, машинне навчання, автокодувальник, застосунок, мультиплатформа.

ЗМІСТ

ВСТУП

Сучасне суспільство характеризується зростанням уваги до здорового харчування, персоналізації дієтичних рекомендацій та оптимізації процесу планування раціону. За даними досліджень, понад 70% користувачів мобільних застосунків виявляють інтерес до персоналізованих рекомендацій щодо харчування, проте традиційні методи підбору страв часто не враховують індивідуальні вподобання, дієтичні обмеження та культурні особливості користувачів [1].

Рекомендаційні системи є одним з найбільш успішних застосувань методів машинного навчання в сучасній індустрії [2]. Вони широко використовуються в електронній комерції (Amazon, eBay), стрімінгових сервісах (Netflix, Spotify), соціальних мережах (Facebook, Instagram) та інших галузях. Застосування рекомендаційних систем у галузі харчування має свою специфіку: необхідність врахування багатовимірних характеристик страв (інгредієнти, харчова цінність, смакові профілі, тип кухні), складність представлення кулінарних даних для машинного навчання. До того ж, все гостріше постає питання захисту користувацьких даних, а тому постає потреба в забезпеченні роботи системи безпосередньо на мобільному пристрої без постійного підключення до мережі Інтернет [3].

Об'єктом дослідження є персоналізовані рекомендаційні системи на користувацьких пристроях.

Предметом дослідження виступають методи побудови компактних векторних представлень мультимодальних даних страв та алгоритми їх ефективної обробки на мобільних пристроях.

Метою роботи є підвищення ефективності та приватності рекомендаційних систем шляхом розробки методу виведення висновків на пристрої користувача.

Для досягнення поставленої мети необхідно розв'язати наступні завдання:

- провести аналіз існуючих підходів та методів машинного навчання для побудови рекомендаційних систем;

- визначити ключові фактори, що впливають на персоналізований вибір страв, провести збір та обробку даних;
- розробити алгоритми роботи рекомендаційної системи та створити модель машинного навчання;
- розробити мультиплатформенний мобільний застосунок з виведенням висновків на пристрої;
- протестувати конфігурації моделі, інтегрувати модель та протестувати прототип.

Для вирішення поставлених завдань використовувались наступні методи: аналіз наукової літератури з проблеми дослідження, порівняльний аналіз, статистичне опрацювання даних, математичне моделювання, експериментальна оцінка.

Наукова новизна дослідження полягає у наступному:

- розроблено гібридний підхід до кодування характеристик страв на основі вкладеного представлення;
- удосконалено архітектуру автокодувальника для навчання ефективних представлень на мобільних пристроях;
- удосконалено метод холодного старту для швидкого формування початкового профілю користувача на основі різноманітного набору страв;

Практичне значення дослідження визначається наступним:

- розроблений мобільний застосунок створює персоналізовані рекомендації страв безпосередньо на пристрої без необхідності постійного підключення до Інтернету, що забезпечує конфіденційність даних користувачів та швидкість роботи системи;
- створено застосунок на базі Kotlin Multiplatform, що працює на Android та iOS, з єдиною кодовою базою, що значно скорочує час та вартість розробки порівняно з нативними рішеннями;
- система може бути розширена та адаптована для застосування в ресторанному бізнесі, сервісах доставки їжі, дієтичному плануванні, освітніх платформах з кулінарії;

- створений прототип демонструє практичне застосування сучасних технологій машинного навчання (TensorFlow, TensorFlow Lite) для мобільних платформ та може слугувати основою для подальших досліджень у галузі персоналізованих рекомендацій.

РОЗДІЛ 1. ОГЛЯД ПІДХОДІВ ТА ТЕХНОЛОГІЙ

1.1 Огляд рекомендаційних систем

Рекомендаційні системи є підкласом систем фільтрації інформації, що прогнозують оцінку або вподобання користувача щодо певного об'єкта. За останні два десятиліття рекомендаційні системи стали невід'ємною частиною сучасних веб- сервісів та мобільних застосунків, допомагаючи користувачам орієнтуватися в величезних обсягах доступної інформації та знаходити релевантний контент [2].

Рекомендаційні системи можна поділити на кілька основних категорій залежно від підходу до генерації рекомендацій [4]:

- фільтрація на основі вмісту – підхід, що базується на аналізі характеристик об'єктів та профілю користувача. Система рекомендує об'єкти, схожі на ті, які користувач оцінював позитивно в минулому [5];
- колаборативна фільтрація – підхід, що базується на аналізі поведінки та вподобань схожих користувачів. Основна ідея: якщо користувачі А та В мають схожі вподобання, то об'єкти, які сподобалися користувачу А, ймовірно сподобаються і користувачу В. Основною перевагою колаборативної фільтрації є можливість знаходити несподівані рекомендації та відкривати нові інтереси користувачів, однак система потребує великої кількості оцінок для ефективної роботи та страждає від проблем холодного старту для нових користувачів та об'єктів [6];
- фільтрація на основі знань (обмежень) – підхід, що використовує явні знання про предметну область, потреби користувачів та властивості об'єктів для генерації рекомендацій. Цей підхід особливо корисний для областей, де користувачі рідко взаємодіють з системою (наприклад, покупка автомобіля чи нерухомості), або коли важливо враховувати специфічні обмеження та вимоги [7];
- демографічна фільтрація – підхід, що використовує демографічну інформацію про користувачів (вік, стать, освіта, місце проживання) для класифікації користувачів та надання рекомендацій на основі демографічних класів. Цей

підхід має обмежену персоналізацію, але може бути ефективним для нових користувачів, коли немає історії взаємодії [8];

– гібридні підходи – комбінація різних методів рекомендацій для подолання недоліків окремих підходів.

Фільтрація на основі вмісту базується на аналізі характеристик (атрибутів) об'єктів та профілю вподобань користувача. Основна ідея полягає у рекомендації об'єктів, схожих на ті, які користувач оцінював позитивно в минулому. На відміну від колаборативної фільтрації, цей підхід не потребує інформації про інших користувачів та їх вподобання [5].

Архітектура системи на основі вмісту складається з трьох основних компонентів [5]:

– аналіз вмісту – відповідає за обробку та структуроване представлення характеристик об'єктів у формі, придатній для наступного етапу обробки;
– навчання профілю – будує та оновлює профіль користувача на основі об'єктів, з якими користувач взаємодіяв, та їх оцінок. Профіль користувача зазвичай представляється у тому ж вигляді, що й об'єкти;
– фільтрація – порівнює представлення об'єктів з профілем користувача та генерує рекомендації на основі міри схожості.

Вибір методу представлення об'єктів є критичним для ефективності системи на основі вмісту [5]. Векторне представлення є класичним підходом, особливо для текстових даних [9]. Кожен об'єкт представляється вектором в n-вимірному просторі, де n – кількість унікальних ознак (термінів). Вага кожної ознаки зазвичай обчислюється за допомогою схеми TF-IDF (Term Frequency - Inverse Document Frequency):

$$w(t,d) = tf(t,d) \times idf(t),$$
$$idf(t) = log(N/df(t))$$

де $tf(t,d)$ – частота терміна t в документі d, N – загальна кількість документів, $df(t)$ – кількість документів, що містять термін t. Це надає більшу вагу термінам,

які часто зустрічаються в конкретному документі, але рідко в усьому наборі, що дозволяє виділити найбільш характерні ознаки об'єкта [9].

Побудова профілю користувача зазвичай здійснюється одним з наступних методів [10]:

– зважене середнє – профіль користувача обчислюється як зважене середнє векторів об'єктів, з якими користувач взаємодіяв, де ваги відповідають оцінкам користувача. Цей простий підхід ефективний, але не враховує зміну вподобань користувача з часом;
– класифікація – профіль користувача представляється класифікатором, навченим на позитивних (сподобалися) та негативних (не сподобалися) прикладах. Популярні методи включають наївний баєсів, дерева рішень та опорних векторів. Класифікатор приймає на вхід вектор характеристик об'єкта та передбачає, чи сподобається він користувачу [11];
– метод релевантного зворотного зв'язку, запозичений з інформаційного пошуку, ітеративно оновлює профіль користувача на основі його оцінок, збільшуючи ваги термінів з позитивно оцінених об'єктів та зменшуючи ваги з негативних [12];

Обчислення схожості між профілем користувача та об'єктами виконується за допомогою метрик схожості векторів. Косинусна подібність є найбільш поширеною метрикою, що обчислює косинус кута між двома векторами:

$$sim(u, i) = cos(\theta) = (u \cdot i) \,/\, (||u|| \times ||i||)$$

де u – вектор профілю користувача, i – вектор об'єкта. Значення варіюється від -1 до 1, де 1 означає повну схожість. Евклідова та Манхеттенська відстані також використовуються для вимірювання близькості у векторному просторі [9].

Переваги фільтрації на основі вмісту [5]:

– не потрібні дані про поведінку інших користувачів;
– прозорість та пояснюваність рекомендацій, можна пояснити, чому об'єкт рекомендовано;

– відсутність проблеми холодного старту для нових об'єктів, потрібен лише опис об'єкта;
– здатність рекомендувати непопулярні, унікальні об'єкти.

Недоліки фільтрації на основі вмісту [5]:

– обмежена різноманітність рекомендацій – система рекомендує лише об'єкти, схожі на вже оцінені;
– проблема холодного старту для нових користувачів, потрібна початкова історія вподобань;
– необхідність значних зусиль для створення якісного опису об'єктів.

Підхід на основі вмісту є природним вибором для кулінарних рекомендацій, оскільки страви мають багатий набір структурованих атрибутів: список інгредієнтів, харчова цінність, час приготування, тип кухні, смакові характеристики [3]. Система може рекомендувати страви на основі схожості інгредієнтів, схожих смакових профілів або дотримання дієтичних обмежень без необхідності збору великої бази оцінок від користувачів. Це робить цей підхід особливо вдалим для мобільних застосунків з обмеженою кількістю користувачів та за умови виведення висновків на пристрої користувача, коли доступ до даних інших користувачів відсутній.

Підходи глибокого навчання до рекомендацій набули значної популярності з розвитком обчислювальних потужностей та доступності великих наборів даних. Нейронні мережі дозволяють автоматично навчати складні нелінійні представлення користувачів та об'єктів, що важко виразити традиційними методами [13].

Автокодувальники використовуються для навчання вкладень у низьковимірному просторі [14]. Кодувальник стискає вхідний вектор ознак в компактне представлення, а декодувальник намагається відновити оригінальний вектор. Навчені вкладення можуть використовуватися для обчислення схожості між об'єктами.

Рекурентні нейронні мережі, зокрема LSTM, використовуються для моделювання послідовностей взаємодій користувача з системою, враховуючи

тимчасову динаміку вподобань. Це особливо корисно для сесійних рекомендацій, коли потрібно передбачити наступну дію користувача на основі поточної сесії [15].

Згорткові нейронні мережі застосовуються для обробки візуального контенту (зображення страв, товарів) та текстових описів. У контексті кулінарних рекомендацій вони можуть аналізувати фотографії страв для визначення їх категорії, інгредієнтів або візуальної привабливості [16].

Нейроколаборативна фільтрація замінює традиційне матричне розкладання для моделювання взаємодій користувачів та об'єктів глибокими нейронними мережами. Замість простого скалярного добутку векторів користувача та об’єкта, вона використовує багатошарові нейронні мережі для навчання складної функції взаємодії [17].

Графові нейронні мережі моделюють користувачів та об'єкти як вузли графа, де ребра представляють взаємодії. Вони можуть враховувати складні взаємозв'язки між об'єктами, наприклад, соціальні зв'язки між користувачами або ієрархію категорій об'єктів [18].

Вкладення є ключовим поняттям у підходах до рекомендацій з використанням глибокого навчання. Замість ручного створення ознак, нейронні мережі автоматично навчають низьковимірні щільні представлення для користувачів та об'єктів. Ці вкладення кодують семантичну схожість: схожі об'єкти мають близькі вектори у векторному просторі.

Переваги підходів з використанням глибокого навчання [13]:

– автоматичне навчання представлень;
– можливість моделювання нелінійних залежностей;
– ефективна робота з неоднорідними даними (текст, зображення, аудіо);
– висока точність на великих наборах даних;

Недоліки підходів глибокого навчання [13]:

– потреба у великих обсягах даних для навчання;
– високі обчислювальні вимоги;
– складність інтерпретації результатів (чорна скринька);
– необхідність налаштування гіперпараметрів для тренування.

Гібридні підходи поєднують переваги різних методів рекомендацій для подолання недоліків окремих підходів. Стратегіями гібридизації можуть бути [4]:

– зважена - зважене об'єднання оцінок від різних методів;
– перемикна - перемикання між методами залежно від ситуації;
– змішана - представлення рекомендацій від різних методів одночасно;
– каскадна -послідовне застосування методів з фільтрацією.

Сучасні рекомендаційні системи часто використовують комбінацію декількох підходів для досягнення кращих результатів. Вибір конкретного підходу залежить від специфіки предметної області, доступності даних, вимог до якості рекомендацій та обмежень системи [2]. Гібридні системи широко використовуються у промислових рішеннях, таких як Netflix [19] та YouTube [20], де комбінація різних методів дозволяє досягти кращої якості рекомендацій.

Проблема холодного старту є однією з найбільш критичних у рекомендаційних системах та виникає, коли система не має достатньо інформації для генерації персоналізованих рекомендацій. Виділяють три основних типи холодного старту: холодний старт користувача – найпоширеніший випадок, коли новий користувач приєднується до системи без історії взаємодій, система не знає його смакових переваг, улюблених кухонь або інгредієнтів; холодний старт елемента виникає, коли в систему додається новий об’єкт без рейтингів або взаємодій від користувачів; холодний старт системи – стартова фаза системи, коли немає ні користувачів, ні взаємодій [6].

Оцінка якості рекомендаційних систем є комплексним завданням, що охоплює декілька вимірів: точність прогнозування, релевантність ранжування, різноманітність та новизну рекомендацій, а також суб'єктивне задоволення користувачів. Відповідно до класифікації, запропонованої в [2], методи оцінки поділяються на три основні категорії: офлайн-експерименти, дослідження за участю користувачів та онлайн-експерименти. Офлайн-метрики базуються на історичних даних взаємодій користувачів з системою та не потребують реальних користувачів для проведення експериментів. До основних офлайн-метрик належать: метрики точності прогнозування (абсолютна похибка,

середньоквадратична похибка), що вимірюють відхилення прогнозованих оцінок від реальних; метрики якості ранжування, що оцінюють здатність системи розмістити релевантні об'єкти на верхніх позиціях списку рекомендацій [2]; метрики різноманітності та покриття, що вимірюють здатність системи рекомендувати широкий спектр об'єктів [28]. Дослідження за участю користувачів передбачають залучення групи учасників для оцінки якості рекомендацій через анкетування, інтерв'ю або контрольовані експерименти. Ці методи дозволяють оцінити суб'єктивні аспекти якості: сприйману корисність рекомендацій, довіру до системи, задоволеність користувацьким досвідом та готовність продовжувати використання системи [32]. Herlocker et al. підкреслюють, що суб'єктивна оцінка користувачів часто не корелює безпосередньо з офлайн- метриками точності, оскільки користувачі цінують не лише точність, а й пояснюваність, новизну та різноманітність рекомендацій [6]. Онлайн- експерименти проводяться на реальних користувачах в робочому середовищі та вимірюють фактичну поведінку: конверсію, час взаємодії з рекомендаціями, утримання користувачів. Онлайн-метрики є "золотим стандартом" оцінки рекомендаційних систем, оскільки вони відображають реальний вплив системи на бізнес-показники та поведінку користувачів [29].

1.2 Рекомендаційні системи в галузі харчування

Рекомендаційні системи для харчування та кулінарії мають унікальні характеристики та виклики, які відрізняють їх від рекомендаційних систем в інших предметних областях. Ці особливості обумовлені складністю та багатовимірністю кулінарних даних, а також різноманітністю факторів, що впливають на харчові вподобання людей [1]. Страви можна описати множиною різнорідних атрибутів:

- інгредієнти – базова характеристика, що визначає склад страви. Список інгредієнтів може містити від кількох до декількох десятків компонентів, кожен з яких має власні властивості та харчову цінність;

- харчова цінність – кількісні показники: калорійність, вміст білків, жирів, вуглеводів, цукру, клітковини, вітамінів та мікроелементів. Ці характеристики критично важливі для користувачів з дієтичними обмеженнями або цілями здорового харчування;
- смакові характеристики – суб'єктивні властивості смаку: солодкий, кислий, солоний, гіркий, пікантний, м’ясний. Смаковий профіль значною мірою визначає сприйняття страви користувачем [21].
- тип кухні – культурна приналежність страви (італійська, китайська, мексиканська, українська тощо), що відображає кулінарні традиції та регіональні особливості приготування;
- тип страви – класифікація за призначенням: закуски, основні страви, десерти, напої, супи, салати тощо;
- метод приготування – спосіб термічної обробки (смаження, варіння, запікання, тушкування), що впливає на смак, текстуру та харчову цінність;
- час приготування – тривалість процесу, що може бути важливим фактором для користувачів;
- складність приготування – рівень кулінарних навичок, необхідних для приготування страви;
- контекст споживання – пора року, час доби, свято, кількість порцій, соціальний контекст (сімейна вечеря, вечірка) [22].

На сьогоднішній день існує низка комерційних та дослідницьких рекомендаційних систем для харчування та кулінарії, що використовують різні підходи та методології [1]. Yummly є однією з найбільших платформ кулінарних рекомендацій, що використовує гібридний підхід, поєднуючи колаборативну фільтрацію з аналізом інгредієнтів та харчової цінності [23]. Система враховує дієтичні обмеження користувачів та персоналізує результати пошуку на основі історії взаємодії. Allrecipes та Tasty використовують подібні підходи з фокусом на соціальних функціях та користувацькому контенті [24]. Freyne та Berkovsky запропонували систему персоналізованого планування харчування, що враховує харчові вподобання, дієтичні цілі та контекст споживання [25]. Harvey et al.

розробили FoodCAS – контекстно-залежну систему, що адаптує рекомендації залежно від часу дня, погоди, поточного емоційного стану користувача [26]. Toledo et al. розробили систему, що балансує між вподобаннями користувача та харчовою цінністю страв, використовуючи багатоцільову оптимізацію для знаходження компромісу між смаком та корисністю [27]. Більшість існуючих рішень працюють у хмарних середовищах та вимагають постійного підключення до інтернету [1]. Рішення з виведенням висновків на пристрої користувача з використанням мобільних моделей машинного навчання залишаються малодослідженими, що визначає актуальність даної роботи.

На основі аналізу існуючих рішень та специфіки предметної області можна виділити ключові виклики та обмеження, що стоять перед розробниками рекомендаційних систем для харчування [3]:

- як для нових користувачів, так і для нових страв існує проблема холодного старту. Користувачі не готові витрачати багато часу на початкове налаштування профілю, тому необхідні швидкі та інтуїтивні методи збору початкової інформації про вподобання [25]. Підходи на основі вмісту вирішують цю проблему для нових страв, але все ще потребують початкової історії взаємодії користувача [5];
- надмірна персоналізація може призвести до ефекту фільтрувальної бульбашки, коли користувач отримує лише схожі рекомендації. Важливо забезпечити здатність системи пропонувати несподівані, але релевантні варіанти, що розширюють кулінарні горизонти користувача [28];
- обчислення рекомендацій у реальному часі для великої кількості користувачів та об'єктів вимагає ефективних алгоритмів та інфраструктури. Традиційні підходи на основі схожості (косинусної подібності) можуть бути повільними для великих наборів даних [23];
- харчові вподобання є чутливою інформацією, що може розкривати медичні стани, релігійні переконання чи культурну приналежність [30]. Користувачі все більше стурбовані приватністю своїх даних, особливо в контексті здоров'я.

Рішення, де модель працює безпосередньо на пристрої користувача без передачі даних на сервер, є перспективним напрямком для вирішення цієї проблеми [31];

– кулінарні набори даних часто містять неповну, неточну або неструктуровану інформацію [3]. Відсутність стандартизації в описі інгредієнтів, різні одиниці виміру, неточні харчові характеристики ускладнюють побудову надійних моделей;
– користувачі хочуть розуміти, чому система рекомендувала конкретну страву. Для глибоких нейронних мереж забезпечення інтерпретованості рекомендацій є складним завданням [32]. Підходи на основі вмісту мають перевагу в цьому аспекті, оскільки можуть пояснити рекомендацію через схожість атрибутів (інгредієнтів, кухні, харчової цінності) [5];
– врахування динамічного контексту (час доби, погода, наявні інгредієнти в холодильнику, настрій) вимагає складної архітектури системи та додаткових джерел даних. Більшість існуючих рішень або ігнорують контекст, або враховують його лише частково [22].

Подолання цих викликів вимагає комплексного підходу, що поєднує передові методи машинного навчання, ефективні алгоритми та уважне врахування специфіки предметної області.

1.3 Технології виведення висновків машинного навчання на пристрої користувача

TensorFlow Lite є спеціалізованим фреймворком від Google для розгортання моделей машинного навчання на мобільних та вбудованих пристроях з обмеженими обчислювальними ресурсами, пам'яттю та енергоспоживанням. На відміну від повноцінного TensorFlow, призначеного для навчання та виведення висновків на серверах, TensorFlow Lite оптимізований для швидкого та

ефективного виведення висновків безпосередньо на пристрої користувача без необхідності підключення до мережі [33].

TensorFlow Lite складається з двох основних компонентів: конвертер – інструмент для конвертації навчених моделей з форматів TensorFlow, Keras та інших у оптимізований формат TensorFlow Lite (.tflite); інтерпритатор – легке середовище виконання для виведення висновків моделей на кінцевих пристроях. Інтерпритатор забезпечує мінімальний розмір (близько 300 КБ для основних операцій) та швидкий запуск, що критично важливо для мобільних застосунків [31].

TensorFlow Lite використовує FlatBuffers – ефективний формат серіалізації даних, розроблений Google. Цей формат не потребує десеріалізації перед використанням, що дозволяє моделям завантажуватися миттєво. Файл .tflite містить граф обчислень моделі, ваги параметрів та метадані у компактному бінарному форматі, оптимізованому для швидкого читання та мінімального споживання пам'яті [33].

Розгортання моделей машинного навчання на мобільних пристроях вимагає їх суттєвої оптимізації через обмеження обчислювальних ресурсів, пам'яті, енергоспоживання та розміру застосунку [31]. TensorFlow Lite застосовує численні оптимізації для покращення продуктивності: об'єднання послідовних операцій в одну для зменшення накладних витрат, попереднє обчислення константних виразів, квантизація параметрів моделі для зменшення розміру та прискорення обчислень, делегування обчислень спеціалізованим акселераторам [34].

Квантизація є одним з найбільш ефективних методів компресії моделей. Квантизація полягає у зменшенні точності представлення параметрів моделі (ваг та активацій) з числових типів високої точності (float32, 32 біти) до типів меншої точності (int8, 8 біт або навіть int4, 4 біти). Це дозволяє зменшити розмір моделі та прискорити обчислення за рахунок використання цілочисельної арифметики, яка виконується швидше на більшості процесорів. Квантизація динамічного діапазону квантизує лише ваги моделі, залишаючи активації в форматі float32, що дає помірне зменшення розміру з мінімальною втратою точності [35].

TensorFlow Lite підтримує делегування обчислень спеціалізованим акселераторам через систему делегатів. На Android доступні GPU делегат - використання графічного процесора для прискорення обчислень, NNAPI делегат – універсальний інтерфейс для доступу до апаратних акселераторів машинного навчання, Hexagon делегат - для чіпів Qualcomm. На iOS підтримується Core ML делегат, що дозволяє використовувати нейронний рушійй в процесорах Apple. Використання акселераторів може прискорити виведення висновків у 5-10 разів порівняно з виконанням на CPU [34].

TensorFlow Lite підтримує широкий спектр платформ: Android (через Java/Kotlin API), iOS (через Swift/Objective-C API), Linux (для вбудованих систем на базі ARM), мікроконтролери (TensorFlow Lite Micro для пристроїв з кілобайтами пам'яті). Кросплатформенна підтримка робить TensorFlow Lite універсальним рішенням для різних типів пристроїв [33].

Серед альтернатив TensorFlow Lite можна виділити: PyTorch Mobile [36] (частина екосистеми PyTorch), ONNX Runtime [37] (кросплатформенне рішення від Microsoft), Core ML [38] (нативний фреймворк Apple для iOS/macOS. Проте TensorFlow Lite залишається найбільш популярним вибором завдяки зрілості інструментарію, широкій підтримці операцій, активній спільноті та безшовній інтеграції з екосистемою TensorFlow [34].

1.4 Мультиплатформенна розробка

Розробка мобільних застосунків для кількох платформ традиційно вимагала створення окремих кодових баз для Android (Java/Kotlin) та iOS (Swift/Objective-C), що призводило до подвоєння зусиль, збільшення часу розробки та складності підтримки. Мультиплатформенні технології дозволяють використовувати спільну кодову базу для різних платформ, значно прискорюючи розробку та зменшуючи вартість створення застосунків [39].

Kotlin Multiplatform є сучасним рішенням від JetBrains для створення мультиплатформенних застосунків з можливістю спільного використання бізнес-логіки між мобільними, веб та настільними платформами. На відміну від інших мультиплатформенних фреймворків, Kotlin Multiplatform не нав'язує єдиний UI фреймворк і дозволяє використовувати нативні інструменти для інтерфейсів на кожній платформі, забезпечуючи повністю нативний користувацький досвід [40].

Kotlin Multiplatform базується на компіляції Kotlin коду в різні цільові платформи: Kotlin/JVM компілює код для Android та серверних застосунків; Kotlin/Native компілює в нативний машинний код для iOS, macOS, Linux, Windows через LLVM; Kotlin/JS компілює в JavaScript для веб-платформ. Спільний код розміщується в модулі commonMain і містить, наприклад, бізнес-логіку, моделі даних, мережеву взаємодію, обробку даних – все, що не залежить від специфіки платформи. Платформо-специфічний код розміщується в окремих наборах вихідного коду (androidMain, iosMain), де реалізуються API, наприклад, для роботи з камерою, сенсорами, системними налаштуваннями тощо [40].

Механізм expect/actual є ключовою особливістю Kotlin Multiplatform, що дозволяє оголошувати платформо-специфічні API у спільному коді. В commonMain оголошується expect декларація (інтерфейс без реалізації), а у платформо- специфічних модулях надаються actual реалізації для кожної платформи. Цей механізм забезпечує типобезпечну абстракцію над платформо-специфічними деталями, дозволяючи спільному коду використовувати платформенні API без прямої залежності від них [40].

Kotlin Multiplatform інтегрований з системою збірки Gradle та підтримує платформо-специфічні залежності через набори вихідного коду. Це дозволяє використовувати як Kotlin Multiplatform бібліотеки (які не використовують платформо-залежні API або вже були адаптовані для Kotlin Multiplatform з використанням expect/actual механізму), так і платформо-специфічні у відповідних модулях [40].

Compose Multiplatform є декларативним UI фреймворком від JetBrains, що розширює Jetpack Compose (який Google презентувала лише на Android) на iOS, веб

та настільні платформи. Compose дозволяє описувати UI декларативно, через композицію функцій, мовою Kotlin. Compose Multiplatform надає можливість спільного використання не лише бізнес-логіки, але й UI коду між платформами, досягаючи рівня спільного коду до 90-95%. Фреймворк рендерить UI через графічний рушій Skia, забезпечуючи однаковий вигляд на всіх платформах, проте зберігає можливість створення платформо-специфічних UI компонентів через expect/actual механізм [41].

Існують альтернативні мультиплатформенні технології, які використовують інші підходи до розробки для багатьох платформ. React Native [42] використовує JavaScript/TypeScript та React для створення мобільних застосунків, відмальовуючи нативні UI компоненти через нативний міст. Flutter [43] від Google використовує мову Dart та власний графічний рушій. Xamarin [44] (нині .NET MAUI) базується на C# та .NET екосистемі. Ionic та Cordova створюють гібридні застосунки на основі веб-технологій, що працюють у нативних компонентах веб-переглядачів на кожній з платформ [45]. Кожна технологія має свої переваги: React Native має велику спільноту та екосистему пакетів, Flutter відомий швидким рендерингом та гарним UI, Xamarin інтегрований з Microsoft екосистемою [39].

Перевагами Kotlin Multiplatform є максимальний відсоток спільного коду, нативна продуктивність (код компілюється в нативний для кожної платформи), можливість збереження нативного UX (через використання платформенних UI фреймворків), можливість поступової міграції (можна поступово додавати Kotlin Multiplatform модулі до існуючих застосунків), активна підтримка JetBrains та зростаюча екосистема бібліотек [40].

Вибір Kotlin Multiplatform для даної роботи обумовлений кількома факторами: можливість нативної інтеграції з TensorFlow Lite на обох мобільних платформах; використання мови Kotlin та її переваг; можливість використання Compose Multiplatform для створення єдиного UI для обох платформ, забезпечуючи консистентний користувацький досвід та прискорюючи розробку інтерфейсу [41].

Висновки до розділу 1

У першому розділі проведено аналітичний огляд предметної області та існуючих рішень. Розглянуто класифікацію рекомендаційних систем, що включає три основні підходи: колаборативну фільтрацію, фільтрацію на основі контенту та гібридні методи. Встановлено, що для задачі персоналізації харчування найбільш доцільним є підхід на основі контенту з використанням глибокого навчання, оскільки він не потребує великої бази користувачів та ефективно працює з багатовимірними характеристиками страв. Досліджено специфіку кулінарних рекомендаційних систем, яка полягає у необхідності врахування інгредієнтів, харчової цінності, смакових профілів, типів кухонь та способів приготування. Проаналізовано існуючі рішення та виявлено їх основні обмеження: залежність від серверної інфраструктури, відсутність автономної роботи та питання конфіденційності даних користувачів. Проаналізовано технології виведення висновків на мобільних пристроях, зокрема бібліотеку TensorFlow Lite та методи оптимізації моделей через квантизацію. Досліджено можливості мультиплатформенної розробки з використанням технології Kotlin Multiplatform, що дозволяє створювати застосунки зі спільною кодовою базою. На основі проведеного аналізу сформульовано вимоги до системи та обґрунтовано вибір технологій для розробки.

РОЗДІЛ 2. РОЗРОБКА СИСТЕМИ ТА ПРОТОТИПУ

2.1 Загальна архітектура системи

Розроблена рекомендаційна система для підбору страв складається з двох основних компонентів: серверного компоненту для навчання моделі машинного навчання та клієнтський компонент, у вигляді мобільного застосунку, для надання персоналізованих рекомендацій користувачам. Така архітектура дозволяє відокремити ресурсоємний процес навчання від процесу виведення висновків, що виконується безпосередньо на пристрої користувача.

Серверний компонент відповідає за підготовку даних, навчання моделі та її оптимізацію для мобільних пристроїв. Цей компонент реалізований мовою Python з використанням бібліотек TensorFlow/Keras та виконується одноразово або періодично при оновленні бази страв. Робочий процес компонента включає наступні етапи:

– набір страв завантажується з JSON файлів, кожен з яких містить структуровану інформацію про страву. Здійснюється аналіз та очищення даних: видалення дублікатів, обробка відсутніх значень, стандартизація назв інгредієнтів;
– для категоріальних ознак будуються словники, що відображають унікальні значення у числові індекси [9]. Створюються словники для інгредієнтів, типів кухонь та типів страв;
– характеристики страв перетворюються у формат, придатний для навчання нейронної мережі. Здійснюється кодування категоріальних ознак для перетворення їх у чисельний вектор. Числові характеристики нормалізуються для забезпечення однакового масштабу [46];
– навчання моделі. Серцем системи є автокодувальник – нейронна мережа, що навчається стискати інформацію про страву в компактне вбудоване представлення та відновлювати вихідні характеристики. Перша частина, кодувальник, приймає на вхід усі підготовлені характеристики страви та генерує низьковимірне представлення, що кодує найважливіші характеристики страви.

Декодувальник намагається відновити вихідні характеристики з вкладення. Модель навчається без учителя, мінімізуючи похибку реконструкції (середньоквадратичну похибку) [47]. Після навчання кодувальник використовується для генерації вкладень всіх страв у наборі. Ці вкладення формують семантичний простір, де схожі страви мають близькі вектори, що дозволяє обчислювати схожість через косинусну подібність [9];

– модель (навчений кодувальник) конвертується у формат TensorFlow Lite;
– для швидкого формування початкового профілю нового користувача система обирає різноманітний набір страв (10-15 штук). Обрані страви зберігаються у JSON файл для використання в мобільному застосунку.

Клієнтський компонент реалізує виведення висновків моделі, обчислення рекомендацій та користувацький інтерфейс. Робочий процес компонента включає наступні етапи:

– при першому запуску мобільного застосунку завантажуються дані страв, модель, метадані та словники; обчислюються вкладення для всіх страв за допомогою моделі; вкладення зберігаються в базі даних для швидкого доступу. Новому користувачу показуються попередньо обрані різноманітні страви, він оцінює кожну страву (подобається/не подобається). Формується початковий профіль користувача як зважена сума вкладень страв, що сподобалися, з негативними вагами для страв, що не сподобалися;
– обчислюється косинусна подібність між профілем користувача та вкладеннями всіх страв у базі даних, найбільш релевантна страва відображається користувачу;
– при кожній взаємодії користувача профіль оновлюється. Вкладення нових страв, на які залишив реакцію користувач, додаються до профілю.

Така архітектура забезпечує повну автономність мобільного застосунку: всі обчислення виконуються локально, не потрібне підключення до Інтернету для отримання рекомендацій, дані користувача залишаються на його пристрої, що гарантує приватність. Водночас, серверний компонент може бути використаний для

періодичного оновлення моделі при появі нових страв у наборі або для тестування різних конфігурацій моделі.

2.2 Підготовка набору даних

Вибір набору даних для навчання рекомендаційної системи є критичним фактором, що визначає якість рекомендацій та можливості системи [3]. Для даної роботи було визначено ряд обов'язкових критеріїв, яким повинен відповідати набір даних страв:

- наявність структурованого списку інгредієнтів для кожної страви. Оскільки система базується на підході на основі вмісту, інгредієнти повинні бути чітко виокремлені;
- наявність категоріальних ознак про страву: тип кухні (італійська, китайська, мексиканська тощо), тип страви (закуска, основна страва, десерт), тип дієти. Ці метадані дозволяють збагатити представлення страви та надати користувачу додаткові можливості для фільтрації [3];
- наявність стабільних посилань на зображення страв, розміщених на віддаленому сервері. Зображення є важливою частиною користувацького досвіду мобільного застосунку, проте збереження тисяч зображень в ресурсах застосунку неприйнятно збільшило б його розмір, а використання власного сервера для розміщення зображень потребувало б значних ресурсів та інфраструктури і виходить за рамки даної роботи;
- наявність харчової цінності: калорійність, кількість білків, жирів, вуглеводів, цукру, клітковини. Ці числові характеристики важливі як для навчання моделі, так і для користувачів, що стежать за своїм раціоном [21];
- наявність посилання на джерело (оригінальний рецепт). Посилання на оригінальну сторінку рецепту дозволяє користувачу отримати детальні інструкції приготування, додаткову інформацію про страву, коментарі інших

користувачів, а також забезпечує належну згадку автора рецепту, що є важливим з точки зору етики та авторського права;

– достатній обсяг набору. Щонайменше декілька тисяч страв для навчання нейронної мережі та забезпечення різноманітності рекомендацій [3].

Було проаналізовано низку публічно доступних наборів даних кулінарних рецептів. Recipe1M/Recipe1M+ [48] є одним з найбільших наборів даних, що містить понад 1 мільйон рецептів з зображеннями, проте він орієнтований на задачу знаходження зв'язку між зображенням та рецептом і не містить структурованої інформації про харчову цінність та метадані в зручному форматі. Набір даних Epicurious [49] містить близько 20 тисяч рецептів з детальними інструкціями та рейтингами, але також не містить стандартизованої харчової цінності, інгредієнтів та посилань на зображення. Food Ingredients and Recipe Dataset with Images [50] орієнтований на розпізнавання інгредієнтів з зображень страв, проте зображення не розміщені на сервері. Серед усіх розглянутих варіантів найбільш відповідним виявився Yummly 28K [51] – набір даних страв від популярного кулінарного агрегатора та рекомендаційного сервісу Yummly [23]. Цей набір повністю задовольняє всім визначеним критеріям: інгредієнти виокремлені; посилання зображень вказують на стабільні сервери; метадані про кухню та тип страви присутні; розрахована харчова цінність; наявне посилання на джерело рецепту; набір страв має достатній обсяг (28 231 рецепт) для навчання моделі та забезпечення різноманітності рекомендацій [3].

Набір даних організований у вигляді 28 тисяч JSON файлів. Типова структура файлу містить наступні поля: id – унікальний ідентифікатор страви; name – назва страви; ingredientLines – масив інгредієнтів у текстовому форматі; nutritionEstimates

– масив об'єктів з харчовою цінністю (білки, жири, вуглеводи, калорії, мікроелементи); flavors – об'єкт з 6 смаковими профілями (солоний, солодкий тощо) зі значеннями від 0 до 1; attributes – об'єкт з метаданими: cuisine (масив типів кухні), course (масив типів страви); images – масив об'єктів з посиланнями на зображення різних розмірів; source – об'єкт з інформацією про джерело рецепту; totalTime – загальний час приготування у текстовому форматі; totalTimeInSeconds – час

приготування у секундах; numberOfServings – кількість порцій; rating – рейтинг страви від користувачів Yummly (1-5).

Проте не в кожному файлі заповнені всі поля: деякі страви можуть не мати зображення, інформації про кухню, харчової цінності або смакового профілю. На основі детального аналізу набору Yummly 28K була визначена еталонна структура даних страви, яка містить всю необхідну інформацію для навчання рекомендаційної системи та відображення страви в мобільному застосунку. Оскільки оригінальний формат набору страв Yummly є надлишковим (містить 74 параметри харчової цінності, включаючи амінокислоти та мікроелементи) та неоднорідним (не всі поля заповнені в усіх файлах), виникла необхідність у трансформації та фільтрації даних. Було розроблено спрощену схему представлення страви з такими обов'язковими полями: id – унікальний ідентифікатор (рядок); name – назва страви (рядок); ingredients – масив інгредієнтів (масив рядків); energyKcal, proteinGram, fatGram, carbohydrateGram, sugarGram, fiberGram – основні показники харчової цінності (дробові числа); sweet, sour, salty, bitter, meaty, piquant – смаковий профіль (дробові числа, діапазон 0-1); cuisine – масив типів кухні (масив рядків); course – масив типів страви (масив рядків); imageUrl – посилання на зображення страви розміром 360 пікселів (рядок); sourceUrl – посилання на оригінальний рецепт (рядок); totalTimeSecond – час приготування в секундах (ціле число). Така структура зберігає всю критично важливу інформацію, відкидаючи надлишкові деталі, що спрощує обробку даних та зменшує розмір датасету.

Для автоматизації процесу трансформації та фільтрації було розроблено скрипт yummlycleaner, мовою Python, що виконує багатоетапну валідацію та перетворення даних. Скрипт читає черговий JSON файл з оригінального набору, для кожного файлу виконує валідацію наявності обов'язкових полів та трансформує валідні дані в еталонний формат. Важливою особливістю скрипта є перевірка актуальності посилань, оскільки ми не володіємо ресурсами, на які посилаємось. Для кожної страви, що пройшла попередню валідацію, виконуються HTTP HEAD запити до посилань на зображення та джерело рецепту, страва включається в очищений датасет лише якщо обидва запити повертають статус код 200 (OK). Процес валідації

та очищення значно скоротив обсяг набору даних: з початкових 28 тисяч лише трохи більше 3 тисяч страв пройшли всі перевірки та були включені в фінальний набір. Проте це є цілком достатнім обсягом для системи на основі вмісту, оскільки кожна страва представлена багатовимірним вектором характеристик та всі документи у фінальному датасеті мають гарантовано високу якість [5].

Після очищення та валідації набору даних страв виникла критична проблема різнорідності представлення інгредієнтів. Типовий рядок з масиву ingredients в наборі даних містить не лише назву інгредієнта, але й кількість, одиниці виміру, спосіб обробки та додаткові коментарі. Наприклад, "4-6 good size chicken breasts (or your favorite chicken parts)" описує інгредієнт курятину, проте для навчання моделі рекомендацій бажано представляти його спрощено як "chicken", щоб система могла легко виявити, що дві різні страви містять однаковий інгредієнт. Без уніфікації, словник інгредієнтів містив би тисячі унікальних рядків, більшість з яких описують той самий базовий інгредієнт у різних формах, що призвело б до розрідженості даних та неможливості виявлення схожості між стравами [52].

Уніфікацію інгредієнтів було вирішено проводити на базі еталонного списку стандартизованих назв інгредієнтів. За основу було взято список з TheMealDB API [79], що містить більше тисячі кулінарних інгредієнтів. Скрипт працює за наступним алгоритмом: завантажує еталонний список інгредієнтів з файлу; для кожного JSON файлу з очищеного набору страв зчитує масив інгредієнтів; для кожного рядка інгредієнта виконує пошук збігів в еталонному списку; на основі знайдених збігів приймає рішення про заміну рядка інгредієнта на еталонний або запитує користувача (ввести заміну вручну або пропустити інгредієнт); зберігає оброблені файли з уніфікованими інгредієнтами.

Після уніфікації всіх інгредієнтів було створено скрипт ingredientscollector, мовою Python, для аналізу частоти використання кожного інгредієнта. Скрипт проходить по всіх файлах страв з уніфікованими інгредієнтами та підраховує скільки разів кожен інгредієнт зустрічається в наборі. Цей аналіз виявив довгий хвіст розподілу: поширені інгредієнти зустрічаються у сотнях страв, тоді як майже сотня інгредієнтів зустрічається лише в 1-5 стравах.

Оскільки завдання системи – виявляти схожість між стравами на основі спільних інгредієнтів, унікальні інгредієнти не можуть сприяти цій меті та не несуть корисної інформації для навчання моделі рекомендацій [9]. Для видалення та заміни рідкісних інгредієнтів було створено скрипт ingredientsoperator, мовою Python, що надає інтерактивний інтерфейс для операцій над інгредієнтами: пошук страв, що містять конкретний інгредієнт; заміна інгредієнта на інший в усіх стравах (заміна на ширшу категорію, що охоплює цей інгредієнт); видалення інгредієнта з усіх страв (якщо інгредієнт є надто специфічним торговим брендом, екзотичним продуктом). Робота зі скриптом була припинена після досягнення мінімального прийнятного результату: всі інгредієнти, що зустрічаються лише в одній страві, були видалені або замінені на ширші категорії. Ця стратегія забезпечила, що кожен інгредієнт у фінальному наборі даних присутній щонайменше у двох стравах, що дозволяє моделі виявляти схожості [9]. Подальше узагальнення менш поширених інгредієнтів (тих, що зустрічаються в 2-10 стравах) могло б додатково покращити якість рекомендацій через зменшення розрідженості представлення інгредієнтів [52].

2.3 Проєктування моделі машинного навчання

Вибір методу рекомендацій для даної системи був обумовлений низкою вимог, що випливають з контексту мобільного застосунку з виведенням висновків на пристрої. Колаборативна фільтрація має критичні обмеження для даного випадку. Цей метод вимагає наявності великої кількості користувачів та їх історій взаємодії зі стравами для побудови матриці взаємодій [6]. Для нового застосунку без існуючої бази користувачів це неможливо. Також колаборативна фільтрація потребує серверного компоненту для збереження та обробки даних усіх користувачів, що суперечить вимозі повної автономності та збереження приватності. Фільтрація на основі знань вимагає складних онтологій та баз знань про харчування, що важко підтримувати та масштабувати [7]. Демографічна фільтрація базується на

демографічних характеристиках користувачів, проте надає занадто загальні рекомендації та не враховує індивідуальні смаки [2]. Фільтрація на основі вмісту є найбільш придатною для даної задачі з кількох причин]: вона базується виключно на характеристиках самих страв, що дозволяє генерувати рекомендації без необхідності в даних інших користувачів; метод ефективно вирішує проблему холодного старту для нових користувачів: достатньо зібрати початкові вподобання через короткий опитувальник, щоб сформувати профіль та почати генерувати персоналізовані рекомендації; цей підхід повністю автономний – після навчання моделі всі обчислення рекомендацій виконуються локально на пристрої користувача без потреби в інтернет-з'єднанні.

Ефективне представлення характеристик страв у форматі, придатному для навчання нейронної мережі, є ключовим фактором якості рекомендацій [9]. Кожна страва в наборі містить різнорідні дані, текстові, категоріальні та числові. Розроблена система підтримує різні методи кодування цих характеристик для максимальної гнучкості та можливості експерименти з різними конфігураціями моделі. Для тренування моделі було створено скрипт train_model мовою Python.

Інгредієнти є найважливішою характеристикою страви для визначення схожості між стравами [52]. Система підтримує два основні підходи до їх кодування:

– кодування з множинною активацією (multi-hot). Спочатку будується словник всіх унікальних інгредієнтів у наборі страв, де кожен інгредієнт отримує унікальний індекс. Залежно від набору даних розмір словника може складати до кількох тисяч. Кожна страва представляється розрідженим вектором розміру словника. Якщо страва містить інгредієнт, відповідна позиція у векторі встановлюється в 1 (або TF-IDF вагу), інакше залишається 0. При використанні TF-IDF, інгредієнти, що зустрічаються в багатьох стравах, отримують низькі ваги, тоді як рідкісні інгредієнти отримують високі ваги, що дозволяє моделі краще диференціювати страви. Перевагою кодування з множинною активацією є інтерпретованість (легко побачити які інгредієнти важливі) та відсутність необхідності навчати додаткові параметри для інгредієнтів. Недоліком є великий

розмір вектору (тисячі елементів) та відсутність семантичної близькості між інгредієнтами [53].

– представлення кожного інгредієнта навченим вкладенням (embedding). В цій конфігурації модель має вхідний шар динамічної форми, що забезпечує можливість обробляти страви з різною кількістю інгредієнтів без доповнення до фіксованої довжини [54]. Кожен індекс проходить через шар вкладення, що відображає його в вектор. Шар вкладення ініціалізується випадково і навчається спільно з усією моделлю, дозволяючи моделі самостійно виявити семантично близькі інгредієнти. Після отримання вкладень для всіх інгредієнтів страви, вони агрегуються в один вектор через операцію об’єднання [55]. Реалізовано два варіанти об’єднання: середнє – обчислення середнього арифметичного вкладень всіх інгредієнтів страви (за допомогою стандартного шару GlobalAveragePooling1D), простий та ефективний метод, що працює добре для більшості випадків; зважене середнє – зважене середнє вкладень, де ваги обчислюються на основі TF-IDF (за допомогою власної реалізації шару TFIDFPooling). Перевагами підходу з вкладеннями є компактність представлення та семантична близькість (модель навчається, що "chicken" і "turkey" схожі). Недоліками є додаткові параметри для навчання та менша інтерпретованість порівняно з кодуванням з множинною активацією.

Подібно до списку інгредієнтів, список типів кухні та список типів страви можуть кодуватись як за допомогою кодування з множинною активацією, так і за допомогою вкладень.

Назва страви може містити корисну інформацію, що не міститься в інших ознаках [56]. Для кодування назв використовується стандартний шар TextVectorization, що виконує токенізацію тексту та побудову словника токенів. Назва перетворюється в послідовність індексів токенів. Кожен токен проходить через шар вкладення, після чого застосовується GlobalAveragePooling1D для отримання фіксованого представлення назви [54]. Використання назв є опціональним, оскільки шар TextVectorization ускладнює конвертацію моделі в

TensorFlow Lite формат (потребує TensorFlow Lite Select ops, що збільшує розмір осередку виконання) [33].

Числові ознаки включають харчову цінність, час приготування та смаковий профіль - загалом 13 числових характеристик. Ці ознаки мають різні діапазони значень: калорії можуть бути від 50 до 2000, час приготування від 300 до 7200 секунд, смакові профілі від 0 до 1. Для забезпечення однакового масштабу всіх числових фічей застосовується стандартизація [46]:

$$x\_norm = (x - \mu) \,/\, \sigma$$

де μ – середнє значення ознаки в наборі даних, σ – стандартне відхилення. Параметри нормалізації (μ, σ) обчислюються один раз на тренувальному наборі даних та зберігаються в метаданих моделі для використання при виведенні висновків. Нормалізація прискорює навчання та покращує його якість [46].

Усі компоненти об'єднуються в модель Keras з множинними входами та одним виходом. Після кодування та нормалізації всіх компонентів, вони конкатенуються в один вектор через шар конкатенації [54]. Результуючий вектор має розмірність, що дорівнює сумі розмірностей всіх компонентів. Об'єднаний вектор проходить через послідовність щільних шарів з ReLU активацією для поступового стискання інформації [13]. Типова конфігурація включає 2-3 шари зі зменшуваною кількістю нейронів, наприклад 256 та 128. Після кожного щільного шару застосовується відсіюючий (dropout) шар для запобігання перенавчанню [54]. Коефіцієнт відсіювання зазвичай встановлюється в діапазоні 0.1-0.3, під час навчання випадково ця частина нейронів виключається з обчислень. Це змушує модель навчатися більш надійним представленням, що не залежать від конкретних нейронів [57]. Останній щільний шар без активації генерує компактне вкладення фіксованої розмірності. Відсутність активації дозволяє вкладенням займати весь простір дійсних чисел, а не обмежуватися позитивним діапазоном (як було б з ReLU) [13].

Фінальним кроком є нормалізація вкладення до одиничної довжини через L2 нормалізацію за допомогою власної реалізації шару L2Normalization. Для обчислення нормалізованого вектора кожен елемент оригінального вектора ділиться на L2 (Евклідову) норму – квадратний корінь суми квадратів елементів вектора. Нормалізація приводить всі вкладення до одиничної гіперсфери, що робить косинусну подібність еквівалентною скалярному добутку. Тепер схожість між двома стравами може бути визначена як скалярний добуток двох векторів, що пришвидшує обчислення.

Вище була описана структура кодувальника. Декодувальник відновлює вихідні характеристики страви з вкладення. Декодувальник має дзеркальну до кодувальника архітектуру, поступово збільшуючи розмірність через щільні шари з ReLU активацією. Ключова ідея автокодувальника: якщо модель може відновити вихідні характеристики з компактного вкладення, значить це вкладення захоплює найважливішу інформацію про страву [13]. Для навчання автокодувальника, кодувальник та декодувальник з'єднуються послідовно: вихід кодувальника (вкладення) подається на вхід декодувальника. Створюється композитна модель, що дозволяє навчати кодувальник та декодувальник спільно через метод зворотного поширення похибки [54]. Декодувальник відновлює лише числові характеристики страв, оскільки вони мають найбільшу інформативність для визначення схожості та не потребують складного декодування [47].

Базова конфігурація відновлює 13 числових ознак. Для цього використовується середньоквадратична похибка:

$$MSE = (1/n)\ \Sigma(y\_true - y\_pred)^2$$

де y_true – справжні значення ознак, y_pred – передбачені значення ознак.

MSE є стандартною функцією втрат для регресійних задач та добре працює для нормалізованих числових даних [54]. Модель мінімізує квадратичну різницю між оригінальними та відновленими значеннями кожної числової ознаки.

При використанні кодування з множинною активацією використовується додатковий шар для відновлення вектора. Сигмоїдна активація ($1 / (1 + e^{-x})$) виводить значення в діапазоні [0, 1], що відповідає ймовірності присутності кожного інгредієнта (типу кухні, типу страви) [54]. Для цієї конфігурації використовується комбінація функцій втрат: середньоквадратична похибка (MSE) для числових ознак та бінарна перехресна ентропія (BCE) для категоріальних:

$$BCE = -\Sigma(y\_true \times log(y\_pred) + (1 - y\_true) \times log(1 - y\_pred))$$

Загальна втрата обчислюється як зважена сума:

$$Total\ Loss = w\_numeric \times MSE + w\_ing \times BCE\_ing + w\_kit \times BCE\_kit + w\_type \times BCE\_type$$

де w_numeric = 1.0, w_ing = 0.5, w_kit = 0.1, w_type = 0.1. Менша вага для категоріальних ознак пов'язана з тим, що їх розмір словника значно більший за кількість числових ознак, і без балансування вони домінували б в похибці. Цей варіант потенційно дозволяє моделі краще навчитися, але значно збільшує кількість параметрів та ускладнює навчання [47].

Для навчання використовується Adam оптимізатор – адаптивний метод градієнтного спуску, що підтримує окремі темпи навчання (learning rate) для кожного параметра. Adam автоматично адаптує темп навчання для кожного параметра під час навчання [58]. Інші гіперпараметри навчання: розмір пакету (batch size), кількість циклів навчання (epochs), коефіцієнт відсіювання (dropout rate) [54].

Для оптимізації процесу навчання використовуються дві функції зворотнього виклику:

– EarlyStopping припиняє навчання, якщо похибка не покращується протягом деякої кількості циклів навчання (за замовчуванням 5). Це запобігає перенавчанню та економить час [59]. Встановлення параметра

restore_best_weights в True відновлює ваги моделі з циклу навчання з найкращою похибкою [54].

– *ReduceLROnPlateau* зменшує темп навчання, якщо похибка виходить на плато [54]. Після деякої кількості циклів навчання (за замовчуванням 3) без покращення, темп навчання множиться на деяке число (за замовчуванням 0.5). Зменшення темпу навчання дозволяє моделі краще калібрувати ваги, коли вона наближається до мінімуму функції втрат [58].

Після завершення навчання автокодувальника, кодувальник відокремлюється від декодувальника та зберігається як самостійна модель, що може генерувати вкладення для нових страв без необхідності в декодувальнику. Косинусна подібність є стандартною метрикою для порівняння векторів в задачах відновлення інформації та рекомендаційних систем [9]. Оскільки всі вкладення нормалізовані до одиничної довжини через L2 нормалізацію (||u|| = ||v|| = 1), формула косинусної схожості спрощується:

$$cosine\_similarity(u, v) = u \cdot v$$

Тобто косинусна подібність стає еквівалентною простому скалярному добутку нормалізованих векторів. Це значно прискорює обчислення, оскільки не потрібно обчислювати норми векторів. Скалярний добуток двох векторів розмірності d вимагає лише d множень та d-1 додавань, що є O(d) операцією, дуже ефективною навіть для мобільних пристроїв для нашого набору даних.

Профіль користувача представляється як зважена сума вкладень страв, що сподобалися користувачу:

$$user\_profile = \Sigma(w\_i \times item\_embedding\_i) \;/\; ||\Sigma(w\_i \times item\_embedding\_i)||$$

де w_i – ваги страв (+1 для тих, що сподобались та -1 для тих, що ні). Сума теж нормалізується до одиничної довжини. Для обчислення рекомендацій система

обчислює косинусну подібність між профілем користувача та вкладеннями всіх страв у базі даних:

$$similarity\_i = user\_profile \cdot item\_embedding\_i$$

Страви ранжуються за спаданням схожості, і страва з найвищим значенням рекомендується користувачу.

2.4 Конвертація моделі для виведення висновків на пристрої користувача

Після навчання модель кодувальник зберігається у форматі Keras (.keras), що є стандартним форматом для TensorFlow моделей [54]. Проте для використання моделі на мобільних пристроях необхідна конвертація в формат TensorFlow Lite (.tflite). Основна складність конвертації полягає в тому, що модель кодувальника приймає входи змінної довжини для інгредієнтів, типів кухні та типів страви при використанні вкладених кодувань [54]. Кожна страва може мати різну кількість інгредієнтів (від 3 до 30), типів кухонь (від 1 до 5) та типів страв (від 1 до 3). TensorFlow під час навчання обробляє це через входи змінної довжини, що дозволяють представляти послідовності змінної довжини ефективно [54]. Проте TensorFlow Lite конвертер не підтримує це нативно, що потребує створення обгортки з явними динамічними формами [33]. Створюється спеціальна обгортка DynamicEncoderWrapper на базі tf.Module. Кожен вхід описується через tf.TensorSpec з динамічною формою. Обгортка просто делегує виклик до оригінального кодувальника, передаючи всі входи як список [54].

Якщо модель використовує назви страв через шар TextVectorization, виникає додаткова складність. TextVectorization містить операції обробки тексту (токенізація, приведення до нижнього регістру, видалення пунктуації), що не входять до стандартного набору TensorFlow Lite операцій [33]. Для підтримки таких операцій необхідно увімкнути TF Select операції. Використання TF Select

операцій має кілька недоліків для мобільних застосунків: збільшення розміру моделі через включення додаткових операцій TensorFlow; збільшення розміру середовища виконання; повільніше виведення висновків, оскільки Select операції не так оптимізовані для мобільних пристроїв як вбудовані; неможливість використання деяких апаратних прискорювачів (GPU може не підтримувати всі Select операції) [33].

TensorFlow надає клас tf.lite.TFLiteConverter для конвертації моделей у TensorFlow Lite формат. Конвертер аналізує граф моделі, ідентифікує всі операції, перевіряє їх сумісність з TensorFlow Lite, та генерує оптимізований граф для виконання. Метод TFLiteConverter.convert() виконує фактичну конвертацію моделі: оптимізацію графа, видалення невикористовуваних операцій, об'єднання послідовних операцій, попереднє обчислення константних виразів, перетворення операцій, серіалізацію, генерацію метаданих [33].

Для мобільних пристроїв квантизація є важливою оптимізацією, оскільки дозволяє розміщувати складніші моделі в обмеженій пам'яті та виконувати виведення висновків швидше, що покращує користувацький досвід [34]. TensorFlow Lite підтримує як квантизацію після навчання, що не потребує повторного тренування моделі, так і навчання з симуляцією квантизації, що забезпечує найкращу точність, але потребує змін у процесі тренування. Для даного проєкту використовується перший варіант, оскільки він простий у застосуванні та забезпечує достатню точність для задачі рекомендацій [35].

Базовим типом квантизації в TensorFlow Lite є квантизація динамічного діапазону. Ваги моделі (щільні шари, шари вкладень) конвертуються з float32 у int8 під час конвертації; активації залишаються у float32 під час виведення висновків; перед виконанням операції ваги динамічно деквантуються з int8 назад у float32. Переваги: зменшення розміру моделі приблизно в 4 рази (оскільки ваги становлять більшість параметрів моделі); не потребує репрезентативного набору даних; незначна втрата точності (<1% для більшості моделей); швидка конвертація [36]. Недоліки: виведення висновків залишається у float32, тому прискорення обмежене

(10-20%); не використовує повний потенціал цілочисельної арифметики на мобільних процесорах [35].

Квантизація float16 конвертує всі ваги та активації з float32 у float16 (16-бітні числа з рухомою комою). Переваги: зменшення розміру моделі приблизно в 2 рази порівняно з float32; прискорення виведення висновків в 2-3 рази на GPU та деяких сучасних CPU з підтримкою FP16 [34]; незначна втрата точності (зазвичай <0.5%) через достатній діапазон float16 для більшості нейронних мереж; сумісність з GPU делегацією на Android та iOS [34]. Недоліки: не всі операції підтримують float16 (всеодно використовується float32 для деяких операцій); менше прискорення на CPU порівняно з int8 квантизацією [36].

Найагресивнішим є int8 тип квантизації, що конвертує всі ваги та активації в 8-бітні цілі числа. Принцип роботи: конвертер пропускає невелику підмножину даних через модель; записує мінімальні та максимальні значення активацій для кожного шару; обчислює оптимальні коефіцієнти масштабування (scale) та зміщення (zero-point); ваги та активації конвертуються з float32 у int8 з відповідними масштабними коефіцієнтами та зміщеннями. Переваги: максимальне зменшення розміру моделі; максимальне прискорення виведення висновків (2-4x) через використання цілочисельної арифметики; мінімальне споживання енергії, що критично для мобільних пристроїв; підтримка спеціалізованих прискорювачів (NNAPI на Android, Neural Engine на iOS) [34]. Недоліки: можлива помітна втрата точності (1-5%) через обмежений діапазон int8; потребує зразкового набору даних для калібрування коефіцієнтів [35].

2.5 Алгоритм холодного старту

Оскільки розроблена система базується на підході на основі вмісту, проблема холодного старту стосується виключно нових користувачів, оскільки набір страв сформовано заздалегідь. При першому запуску застосунку користувач має

порожній профіль, тому обчислення косинусної подібності між профілем користувача та вкладеннями страв неможливе.

Існує декілька класичних стратегій боротьби з холодним стартом користувача [60]:

- популярні елементи – показувати найпопулярніші страви на основі глобальної статистики (найбільше лайків, найвищий рейтинг). Переваги: прості в реалізації, гарантовано показують якісний контент. Недоліки: не враховують індивідуальні вподобання, всі нові користувачі бачать однакові страви, не сприяють рекомендаціям менш популярного контенту;
- демографічні рекомендації – використовувати демографічні дані користувача (вік, стать, регіон, освіта) для пошуку схожих користувачів та рекомендації того, що подобається їм. Переваги: враховують певний контекст користувача. Недоліки: потребують збору особистих даних, грунтуються на стереотипах, демографічна подібність не гарантує схожість смаків;
- явний збір вподобань – при першому запуску запитувати користувача про його вподобання через анкету або інтерактивний опитувальник. Типові питання: "Які кухні ви любите?", "Які інгредієнти ви не вживаєте?" (алергії, вегетаріанство), "Ваші улюблені страви?" (вибір з попередньо визначеного списку). Переваги: безпосередньо отримуємо інформацію про вподобання, працює для всіх типів рекомендаційних систем, користувач відчуває контроль над персоналізацією. Недоліки: додаткове тертя при адаптації (користувачі можуть пропустити анкету або покинути застосунок), вимагає зусиль від користувача до того, як він побачить цінність продукту, складно для користувачів, які не знають своїх переваг або не вміють їх артикулювати;
- гібридні підходи – комбінація кількох стратегій для пом'якшення недоліків кожної. Наприклад: показувати популярні страви з різних категорій (італійські, азіатські, десерти) для різноманітності; одночасно збирати неявні сигнали (які страви користувач переглядає довше, які пропускає); поступово формувати профіль на основі перших взаємодій та швидко адаптувати рекомендації .

Для мобільного застосунку з виведенням висновків на пристрої користувача критичними є наступні вимоги: мінімальне тертя при адаптації – користувач повинен якомога швидше побачити цінність застосунку, тому довгі анкети небажані; різноманітність початкових рекомендацій – навіть без профілю користувача, система має показувати різноманітні страви з різних кухонь, категорій та смакових профілів, щоб дати користувачу можливість відкрити для себе нові смаки; локальна обробка – алгоритм холодного старту має працювати повністю на пристрої без потреби в серверних обчисленнях або збереженні даних на сервері.

Для вирішення проблеми холодного старту в даній системі було обрано стратегію вибору різноманітного набору страв для початкового екрану. Замість показу популярних або випадкових страв, система при першому запуску демонструє набір попередньо відібраних страв. Ця стратегія забезпечує: покриття широкого спектру смаків, кухонь, інгредієнтів та харчових профілів; можливість для користувача швидко знайти щось, що йому сподобається, незалежно від його смаків; ефективне формування початкового профілю користувача на основі перших реакцій, оскільки різноманітні страви несуть більше інформації про вподобання [61]. Користувач одразу бачить цікавий контент без необхідності заповнювати анкету, може почати взаємодіяти з застосунком, а після кількох взаємодій система переходить до повноцінних персоналізованих рекомендацій на основі сформованого профілю.

Для забезпечення різноманітності страв для холодного старту було розроблено гібридний алгоритм, що поєднує кілька стратегій для покриття різних аспектів кулінарних вподобань. Алгоритм виконується один раз після навчання моделі та формує фіксований набір страв, що зберігається в мобільному застосунку для показу всім новим користувачам.

Система використовує гібридний підхід з чотирьох етапів, кожен з яких забезпечує різноманітність по різних аспектах:

– відбір за кухнями. Спочатку підраховується частота кожної кухні в наборі страв. Обираються найпоширеніші кухні, що гарантує представлення популярних кулінарних традицій. Для кожної обраної кухні знаходиться перша страва з

набору, що належить до цієї кухні та ще не була обрана на попередніх етапах. Такий підхід забезпечує, що користувач побачить страви різних кулінарних культур, дозволяючи йому виразити вподобання до певних регіональних смаків [61];

– відбір за типами. Аналогічно до кухонь, підраховується частота кожного типу, обираються найпоширеніші типи та для кожного знаходиться представник, що ще не був обраний. Це важливо, оскільки користувачі можуть мати різні вподобання для різних типів страв: хтось може любити італійські десерти, але не любити італійські основні страви;
– відбір за типами інгредієнтів. Визначено шість основних груп інгредієнтів, що охоплюють різні дієтичні та смакові профілі: м’ясо, риба, овочі, бобові, молочні продукти, крупи. Для кожної групи алгоритм шукає першу страву, інгредієнти якої містять хоча б одне ключове слово з групи. Цей етап критичний для виявлення дієтичних обмежень та основних харчових вподобань користувача [52].
– максимальна віддаленість в просторі вкладень. Цей етап використовує косинусну подібність для вимірювання віддаленості між вкладеннями. Алгоритм працює жадібно: обчислюється середнє вкладення всіх вже обраних страв (або всього набору даних, якщо це перша ітерація); для кожної доступної страви (що ще не була обрана) обчислюється косинусна подібність до вже обраних страв; обирається страва з мінімальною подібністю; процес повторюється до досягнення потрібної кількості страв. Цей етап забезпечує, що страви будуть максимально різними, покриваючи різні підкатегорії смаків та стилів приготування [61].

Після завершення відбору, ідентифікатори обраних страв зберігаються в JSON файл та додаються до ресурсів мобільного застосунку, щоб страви для холодного старту були доступні одразу після встановлення без потреби в мережевих запитах. Після того, як користувач взаємодіє зі стравами холодного старту, система формує його початковий профіль: для кожної страви, що отримала позитивний відгук, її вкладення додається до профілю, для кожної страви, що отримала негативний

відгук, її вкладення віднімається з профілю (додається з від'ємною вагою); результуючий вектор нормалізується до одиничної довжини через L2 нормалізацію. Після цього профіль користувача готовий для генерації персоналізованих рекомендацій. Система також підтримує інкрементальне оновлення профілю: при кожній подальшій взаємодії профіль оновлюється через формулу:

$$profil\_new = normalize(profile\_old + \alpha \times weight \times item\_embedding)$$

де α – темп навчання (0.1-0.5).

2.6 Архітектура мобільного застосунку

Створення Kotlin Multiplatform проєкту зручно робити за допомогою веб сервісу [62] що значно спрощує процес створення нового проєкту. Сервіс надає інтерактивний інтерфейс для конфігурації проєкту: вибір цільових платформ (Android, iOS, Desktop, Web, Server); вибір базових бібліотек та фреймворків; конфігурація системи збірки; налаштування структури пакетів та простору імен. Після заповнення форми сервіс генерує повну структуру проєкту з налаштованими конфігураційними файлами, наборами вихідного коду для кожної платформи, базовими класами та прикладами коду [62].

Для створеного застосунку було обрано ім’я Redish. При створенні проєкту RedishApp через згаданий сервіс та подальшій розробці було обрано наступні ключові бібліотеки:

– Compose Multiplatform – мультиплатформенний декларативний UI фреймворк від JetBrains, що дозволяє писати UI код один раз для Android, iOS, Desktop та Web з використанням Kotlin. Переваги: декларативний синтаксис (UI як функція стану), що значно спрощує розробку порівняно з імперативними підходами; реактивне оновлення UI через рекомпозицію; багата екосистема компонентів

Material. Compose Multiplatform було обрано через можливість спільного використання всієї UI логіки між платформами та найкращу інтеграцію з екосистемою Kotlin [41].

– Decompose – бібліотека для навігації та управління життєвим циклом компонентів у Kotlin Multiplatform застосунках. Decompose надає архітектурний паттерн для розділення бізнес-логіки та UI з підтримкою життєвого циклу, збереження стану, глибоких посилань. Decompose дозволяє писати навігаційну логіку в commonMain без дублювання для кожної платформи; забезпечує правильне управління життєвим циклом компонентів; підтримує автоматичне збереження та відновлення стану при змінах конфігурації [62].
– Koin – легкий фреймворк для введення залежностей для Kotlin та Kotlin Multiplatform. Koin використовує власні інструкції для визначення модулів та залежностей без кодогенерації та рефлексії. Koin є найпопулярнішим в своєму класі рішенням з відмінною підтримкою Kotlin Multiplatform (спільні модулі в commonMain, платформо-специфічні в androidMain/iosMain) та активною спільнотою [63].
– Room – ORM бібліотека від Google з експериментальною підтримкою Kotlin Multiplatform. Room надає типобезпечні API для роботи з базою даних SQLite через анотації та кодогенерацію. Room є стандартом індустрії для Android розробки, експериментальна підтримка Kotlin Multiplatform дозволяє використовувати єдину схему для обох платформ, генерує платформо-специфічні реалізації автоматично [64].
– Kotlinx Serialization – офіційна бібліотека для серіалізації/десеріалізації даних у Kotlin. Підтримує JSON, ProtoBuf, CBOR та інші формати через кодогенерацію. Причини вибору: необхідність десеріалізації страв та інших даних з формату JSON, найкраща інтеграція з Kotlin Multiplatform [65].
– Sketch – сучасна бібліотека для завантаження та відображення зображень у Compose Multiplatform. Sketch надає найкращу Compose Multiplatform інтеграцію, підтримує асинхронне завантаження з мережі та кешування [66].

– Arrow – функціональна бібліотека для Kotlin, що надає типи для обробки помилок (Either, Option) та функціональні утиліти. Arrow дозволяє писати більш безпечний код через явну обробку помилок замість виключень. Проєкт використовує Arrow для обробки результатів запитів до бази даних, виведення висновків; явна обробка помилок покращує надійність [67].

Додаткові бібліотеки**:** Kermit – логування для Kotlin Multiplatform [68]; Kotlinx Coroutines – асинхронне програмування [69]; Kotlinx Datetime – робота з датами та часом [70]; Compottie – Lottie анімації для Compose Multiplatform [71]; WebView Multiplatform – відображення веб-контенту [72]. Ці бібліотеки було обрано для покриття типових потреб мобільного застосунку з мультиплатформенним підходом.

Архітектура мобільного застосунку Redish побудована на основі принципів чистої багаторівневої архітектури з чітким розділенням відповідальностей. Код організовано у три основні рівні: рівень предметної області (domain layer), рівень даних (data layer) та рівень презентації (presentation/UI layer), що забезпечує високу підтримуваність та можливість незалежного розвитку кожного рівня[73].

Рівень предметної області містить ядро бізнес-логіки застосунку та є повністю незалежним від деталей реалізації інших рівнів. Структура рівня:

а) domain/contract – інтерфейси бізнес-логіки:

1) DishRepository - інтерфейс для доступу до даних страв;
2) RecommendationService - інтерфейс для генерації рекомендацій;
3) model/Dish - модель страви предметної області;
4) DomainError - клас для типізованих помилок; б) domain/impl – реалізації бізнес-логіки:
1) DishRepositoryImpl - реалізація репозиторію, перетворює сутності рівня даних в моделі предметної області та навпаки;
2) RecommendationServiceImpl - основна логіка рекомендацій;
3) ml/ – ML підсистема:
   – ModelInference - інтерфейс для платформо-специфічного виведення висновків;

- DishFeatureExtractor - перетворення моделі предметної області страви в вид, прийнятний для моделі машинного навчання;
- RecommendationEngine - обчислення схожості, холодний старт, оновлення профілю.

Рівень даних відповідає за збереження даних, завантаження ресурсів та взаємодію зі сховищем даних. Структура рівня:

а) data/contract – інтерфейси для сховища:

1) DishStorage - операції з базою даних страв,
2) UserStorage - збереження профілю, реакцій, налаштувань,
3) model/ – сутності:
   - DishEntity,
   - DishWithRelations,
   - UserData,
   - DatabaseError (типізовані помилки бази даних). б) data/impl – реалізації інтерфейсів:

1) RedishDatabase - Room база даних,
2) DishDao – робота з даними страв в базі даних,
3) UserDao – робота з даними користувача в базі даних ,
4) DishStorageImpl – реалізація інтерфейсу,
5) RoomUserStorage – реалізація інтерфейсу,
6) DatabasePopulator - завантаження страв з JSON при першому запуску,
7) EmbeddingPopulator - збереження вкладень при першому запуску.

Рівень даних використовує Room для Kotlin Multiplatform з анотаціями Entity, Dao, Database для кодогенерації платформо-залежних реалізацій [64]. Рівень предметної області та рівень даних поділені на інтерфейси (contract) та реалізації (implementation), що відповідає принципам чистої архітектури та SOLID [73].

Рівень презентації організовано за компонентами бізнес-логіки, кожна з яких є функціональним блоком застосунку. Типова структура модуля рівня презентації: а) ui/ – Compose UI та Decompose компонент:

1) *Component.kt - Decompose компонент з життєвим циклом та навігацією;

2) *Content.kt - Composable UI функції;
3) *Contract.kt - класи стану (event), подій (event), дій (action);
4) *Instance.kt - бізнес-логіка компонента, взаємодія з рівнем предметної області.

б) *Module.kt – Koin модуль для реєстрації залежностей компонента [63].

Бізнес-компоненти використовують підхід однонаправленого потоку даних (UDF) на базі архітектури MVI: інтерфейс реєструє події від користувача → Instance оброблює події та оновлює стан → інтерфейс реагує на зміни стану. Instance також може ініціювати дії для навігації [74]. Decompose компоненти взаємодіють через чітко визначені контракти [62]: батьківський компонент створює дочірні компоненти та передає в них функції зворотнього виклику для навігації (інверсія залежностей) [73]; Дочірні компоненти викликають ці функції без знання про реалізацію навігації; Decompose автоматично управляє життєвим циклом компонентів: компонент створюється при навігації, зберігає стан при зміні конфігурації, знищується при видаленні зі стеку навігації. При смерті процесу Decompose серіалізує стан навігації через kotlinx.serialization та відновлює всю ієрархію навігації [62]. Якщо представити структуру компонентів у вигляді дерева (Рисунок 2.1), то користувацький інтерфейс мають лише листи дерева. Решта компонентів вирішує задачі навігації, життєвого циклу та передачі даних між дочірніми компонентами.

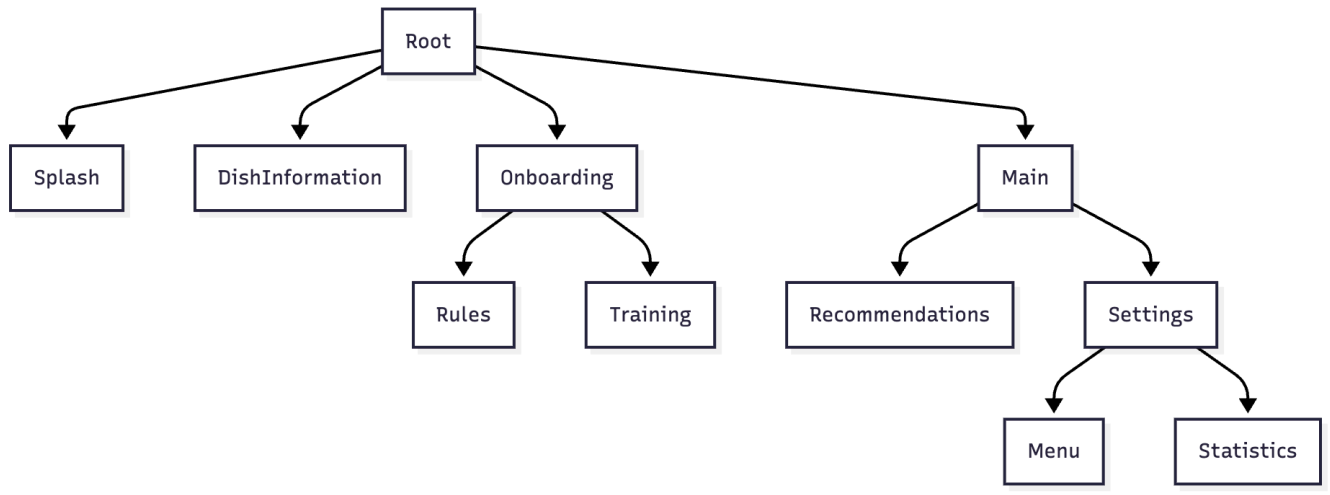

Рисунок 2.1 – Ієрархія компонентів рівня презентації

Ієрархія рівня презентації складається з наступних компонентів:

- кореневий компонент (Root) є точкою входу застосунку та керує глобальною навігацією між основними компонентами;
- компонент завантаження (Splash) відображається при запуску застосунку та виконує ініціалізацію, що супроводжується анімацією завантаження. Він перевіряє, чи користувач завершив анкетування, та здійснює навігацію до компонента анкетування або головний компонент, залежно від результату. При першому запуску компонент завантажує в базу даних страви та обчислює вкладення страв;
- компонент анкетування **(**Onboarding) займається проблемою холодного старту для нових користувачів. Він керує двома дочірніми компонентами: компонентом правил та компонентом тренування;
- компонент правил (Rules) пояснює правила холодного старту та роботу з застосунком, завантажує страви для холодного старту, навігує до компоненту тренування;
- компонент тренування (Training) збирає від користувача реакції на страви холодного старту та оновлює профіль користувача. Навігує до головного компонента після завершення анкетування або до компоненту деталей про страву;
- головний компонент (Main) керує двома основними компонентами застосунку: компонентом рекомендацій та компонентом налаштувань;
- компонент рекомендацій (Recommendations) є ключовим екраном застосунку, що відображає персоналізовані рекомендації страв. Він завантажує рекомендації, збирає реакції він користувача та оновлює профіль користувача. Має кнопку для переходу до компоненту налаштувань та до компоненту деталей про страву;
- компонент налаштувань (Settings) керує двома дочірніми компонентами: компонентом меню та компонентом статистики;
- компонент меню (Menu) надає користувачу можливості очистити дані, переглянути статистику або змінити локалізацію застосунку;

– компонент статистики (Statistics) надає користувачу інформацію про список оцінених страв та його профіль (його вкладення);
– компонент деталей про страву (DishInformation) є модальним екраном для відображення повної інформації про страву, що міститься в системі, включно зі вкладенням. Екран має кнопку перегляду рецепту для відкриття оригінального рецепту в веб-переглядачі.

LazySwipeCards є розробленим Compose віджетом для надання користувачу можливості давати реакцію (позитивну чи негативну) на деякий об’єкт за допомогою змахування картки з цим об’єктом вліво чи вправо. Віджет має систему зворотнього виклику для сповіщень про взаємодію з картками. Стек карток гарно підходить для даної рекомендаційної системи, оскільки ми пропонуємо користувачу найбільш релевантну страву та ненав’язливо просимо його дати реакцію для отримання наступної рекомендації. Цей віджет використовується на екранах компонентів тренування та рекомендацій.

Управління даними в застосунку реалізовано через Room ORM бібліотеку з експериментальною підтримкою Kotlin Multiplatform. Набір даних страв зберігається в SQLite базі даних для ефективного пошуку, фільтрації та вставки [75]. При першому запуску застосунку клас DatabasePopulator завантажує dishes.json з ресурсів та наповнює базу даних. База даних страв складається з 8 сутностей: 5 основних таблиць та 3 єднальні таблиці для відношень багато-до-багатьох між стравами та інгредієнтами/типами/кухнями [76]. Room автоматично генерує ефективний SQL код з правильними JOIN-ами та використовує шаблон DAO для абстракції запитів до бази даних [64]. Структуру таблиць та зв’язків між ними зображено на рисунку 2.2.

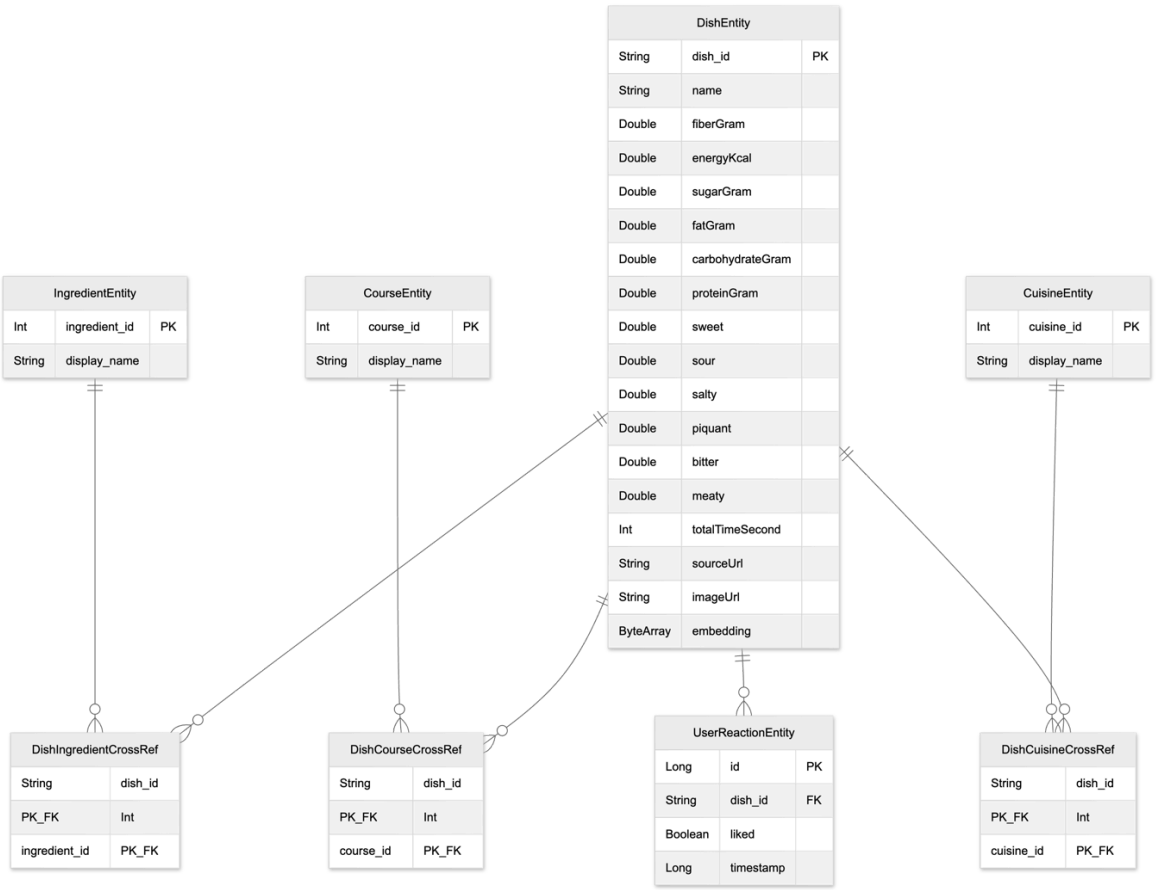

Рисунок 2.2 – структура бази даних

Інтеграція TensorFlow Lite у Kotlin Multiplatform застосунок є нетривіальною задачею, оскільки Android та iOS використовують різні бібліотеки з різними API [33]. У commonMain визначено інтерфейс ModelInference, що представляє єдиний API для виведення висновків на всіх платформах. Методи інтерфейсу: initialize(modelMetadata: ModelMetadata) – ініціалізація інтерпритатора з моделлю та метаданими; computeEmbedding(features: DishFeatures): FloatArray – виконання виведення висновків для генерації вкладення страви; close() – звільнення ресурсів інтерпритатора. Інтерфейс не містить платформо-специфічних типів та може бути викликаний зі спільного коду.

Android реалізація використовує офіційну LiteRT (TensorFlow Lite) бібліотеку для Kotlin/Java [33]. Підключення залежності: у shared/build.gradle.kts в androidMain.dependencies додано три бібліотеки: implementation(libs.litert) (ядро середовища виконання, ~300 KB), implementation(libs.litert.support) (утиліти для підготовки даних), implementation(libs.litert.metadata) (читання метаданих з моделі) [33]. LiteRT є новою назвою TensorFlow Lite після ребрендингу від Google.

TensorFlow Lite модель (model.tflite) розміщується в androidApp/src/main/assets. Interpreter(modelBuffer) створює TensorFlow Lite інтерпритатор. Метод computeEmbedding виконує звернення до інтерпритатора. Створюється масив входів inputs: Array<Any?> з динамічною кількістю входів залежно від конфігурації моделі; якщо використовується ім’я страви то перший вхід є Array<String> з назвою страви; наступні входи: ingredientsMulti (FloatArray) (або ingredientsIdx (IntArray) для вкладень), courseMulti (FloatArray) (або courseIdx (IntArray) для вкладень), cuisineMulti (FloatArray) (або cuisineIdx (IntArray) для вкладень), numeric (FloatArray). runForMultipleInputsOutputs(inputs, outputMap) виконує виведення висновків з багатьма входами та одним виходом. Вихід - FloatArray розміру embeddingDim (64 або 128) [33].

iOS реалізація написана мовою Swift та використовує TensorFlow Lite Swift бібліотеку. Підключення залежності: у iosApp.xcodeproj через Swift Package Manager додано TensorFlowLiteSwift. TensorFlow Lite модель копіюється в iOS app bundle (Resources). Interpreter(modelPath: modelPath, options: options) створює інтерпритатор, до якого звертається метод computeEmbedding. allocateTensors() виділяє пам'ять для входів та виходів моделі (викликається один раз). Метод computeEmbedding має більш складну реалізацію порівняно з Android через особливості TensorFlow Lite Swift API [33]. Фази виведення висновків:

– підготовка даних: конвертація Kotlin типів (KotlinFloatArray, KotlinIntArray) у Swift [Float] та [Int32], створення Data об'єктів;
– зміна розмірів входів: TensorFlow Lite вимагає явного виклику resizeInput(at: index, to: shape) для кожного входу з динамічною довжиною; для ingredientsIdx розмір є count, де count – кількість інгредієнтів; для cuisineIdx та courseIdx – аналогічно; для числових ознак форма фіксована - 13;
– виділення пам’яті: allocateTensors() викликається після всіх змін розмірів для виділення правильної кількості пам'яті;
– копіювання даних: interpreter.copy(data, toInputAt: index) копіює Data у відповідний вхідний тензор; порядок копіювання повинен відповідати порядку входів у моделі [33];

– виведення висновків: interpreter.invoke() виконує звернення до моделі.
– вивід: interpreter.output(at: 0) повертає вихідний тензор; Data конвертується назад у [Float]; результат конвертується у KotlinFloatArray для повернення в Kotlin код.

Для забезпечення виклику платформо-залежної реалізації в спільному коді в цій ситуації неможливо просто використати expect/actual механізм. Реалізація на iOS базується на Swift коді та його бібліотеках, а actual імплементація, на даному етапі розвитку Kotlin Multiplatform, може бути лише мовою Kotlin; виклик Swift коду з Kotlin потребує значних зусиль та роботи з Objective-C заголовками. В нагоді стане те, що інтерфейси Kotlin автоматично експортуються як Objective-C протоколи, що дозволяє Swift класам їх імплементувати [40].

Платформо-залежна реалізація вводиться в загальний код через Koin. Android: в MainActivity створюється AndroidModelInference(context) та передається в Koin при ініціалізації. iOS: в iosApp.swift створюється IosModelInference() та передається в Kotlin функцію initKoin(modelInference: ModelInference), оскільки Swift клас IosModelInference імплементує Kotlin інтерфейс ModelInference (через Objective-C міст); На боці Kotlin initKoin, аналогічно з Android, викликає startKoin та вводить платформенну реалізацію в граф залежностей [63].

RecommendationEngine в спільному коді отримує ModelInference через введення в конструктор за допомогою Koin:
class RecommendationEngine(private val modelInference: ModelInference, ...)
Клас викликає modelInference.computeEmbedding(features) без знання про платформенну імплементацію. Це реалізує повну абстракцію платформо-залежного виведення висновків машинного навчання за єдиним інтерфейсом.

Висновки до розділу 2

У другому розділі розроблено архітектуру персоналізованої рекомендаційної системи для підбору страв. Спроєктовано загальну структуру системи, що складається з двох основних компонентів: модуля машинного навчання та мультиплатформенного мобільного застосунку. Підготовлено набір даних для навчання моделі на основі колекції Yummly28K. Після очищення та стандартизації отримано 3071 страву з повною інформацією. Проведено уніфікацію інгредієнтів: вихідний перелік з понад 6000 унікальних найменувань зведено до 562 стандартизованих інгредієнтів шляхом видалення дублікатів, об'єднання синонімів та нормалізації написання. Спроєктовано архітектуру нейронної мережі-автоенкодера для формування 64-вимірних векторних представлень страв. Розроблено гібридний підхід до кодування характеристик, що поєднує: векторні представлення інгредієнтів розмірністю 32 із середньою агрегацією; шар текстової векторизації для назв страв; 6 числових ознак харчової цінності; 6 ознак смакового профілю; категоріальні ознаки типу кухні та типу страви. Архітектура кодувальника включає два повнозв'язних шари [256, 128] з функцією активації ReLU та регуляризацією виключенням 20% нейронів. Розроблено алгоритм холодного старту для нових користувачів, що базується на методі максимізації різноманітності. Алгоритм обирає 10-15 репрезентативних страв різних кухонь та типів для швидкого формування початкового профілю вподобань. Описано процес конвертації моделі у формат TensorFlow Lite. Спроєктовано архітектуру мультиплатформенного мобільного застосунку на базі Kotlin Multiplatform з використанням бібліотек Compose Multiplatform для інтерфейсу користувача, Decompose для навігації та Room для локальної бази даних. Вихідний код обох компонентів рекомендаційної системи розміщено в репозиторії [80].

РОЗДІЛ 3. ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ

3.1 Тестування конфігурацій моделі

Всі роботи з тренування моделі та розробки мультиплатформенного мобільного застосунку виконувались на MacBook Pro (2021) з наступними характеристиками: процесор Apple M1 Pro з 8-core CPU та 14-core GPU; 16 GB спільної оперативної пам'яті для CPU, GPU та NPU); macOS Sequoia 15.5. Apple M1 Pro має вбудований нейронний рушій для прискорення операцій машинного навчання [77]. MacBook був необхідний для розробки iOS версії застосунку, адже Xcode для компіляції iOS застосунків доступний тільки на macOS. Для тренування використовувався Python 3.10 в віртуальному середовищі venv.

Основна робота по створенню моделі відбувається за допомогою скрипта train_model. Окрім всіх необхідних функцій для завантаження даних, створення, тренування та конвертації моделі та визначення страв для холодного старту скрипт також надає користувачу базовий інтерфейс командного рядка для інтерактивного задання конфігурації моделі та інших налаштувань. При старті train_model, після завантаження страв, скрипт починає запитувати користувача:

- чи враховувати назву страви в моделі;
- який метод кодування категоріальних ознак використовувати: кодування з множинною активацією чи вкладення. В разі обрання вкладень запитуються розміри вкладень та метод їх об’єднання: середнє чи зважене середнє. В разі обрання mutli-hot кодування запитується чи використовувати TF-IDF вагування та розмір щільного шару для інгредієнтів;
- розмір фінального вкладення для страв;
- розмір щільних шарів нейронної мережі;
- коефіцієнт відсіювання;
- темп навчання;
- розмір пакету навчання;
- кількість циклів навчання;

– чи створювати набір страв та для холодного старту та якого розміру;
– чи конвертувати модель в TF Lite формат. В разі позитивної відповіді запитується, чи створювати квантизовану версію та який метод квантизації обрати.

Після виконання скрипт виводить інформацію про фінальний розмір похибки, розмір моделі, кількість параметрів в моделі, Отримані в результаті виконання скрипта файли моделі, словника, метаданих та набору страв для холодного старту копіюються в ресурси застосунку.

Для оцінки впливу різних архітектурних рішень та визначення опитимальної конфігурації для даного проєкту було проведено серію з 10 експериментів на повному наборі страв. Всі експерименти базувались на такій конфігурації: використовувати назву страви, розмір вкладень для інгредієнтів – 32, розмір вкладень для типів страв та кухонь – 8, розмір вкладень для страви – 64, розмір щільного шару для інгредієнтів при кодуванні з множинною активацією – 64, темп навчання - 0.001; розмір пакету - 128 страв, 20 циклів навчання, коефіцієнт відсіювання – 0.2, розміри щільних шарів – 256 та 128. Було здійснено тренування моделей з такими конфігураціями (якщо значення параметра не вказано то воно ідентичне базовій конфігурації), в дужках вказано кодову назву для експерименту:
– кодування з множинною активацією (mutlihot);
– кодування з множинною активацією зі зваженим вагуванням (multihot-tfidf);
– вкладення з усередненням (emb-mean);
– вкладення зі зваженим усередненням (emb-tfidf);
– вкладення зі зваженим усередненням, розмір вкладень для страв - 32 (emb-tfidf-32);
– вкладення зі зваженим усередненням, розмір вкладень для страв - 128 (emb-tfidf-128);
– вкладення зі зваженим усередненням, розмір вкладень для страв - 256 (emb-tfidf-256);
– вкладення зі зваженим усередненням, назва страви не враховується (noname-emb-tfidf);

– вкладення зі зваженим усередненням, розмір вкладень для інгредієнтів - 64 (ing64-emb-tfidf);
– вкладення зі зваженим усередненням, розміри щільних шарів – 512, 256, 128 (emb-tfidf-deep).

Оскільки розмір набору страв відносно невеликий, а Apple M1 Pro має апаратне прискорення для навчання моделей машинного навчання, час тренування моделей виявився вкрай незначним, навчання моделі кожної конфігурації займало до 5 секунд, тому цей критерій при аналізі конфігурацій не враховувався. Результати проведених експериментів наведено в таблиці 3.1.

Таблиця 3.1 – Результати тренувань з різними конфігураціями

Конфігурація	MSE	MAE	Кількість параметрів	Розмір
multihot	0.1330	0.1878	167K	752K
multihot-tfidf	0.1105	0.1892	167K	752K
emb-mean	0.0784	0.1824	146K	671K
emb-tfidf	0.0796	0.1843	146K	671K
emb-tfidf-32	0.0867	0.1878	141K	655K
emb-tfidf-128	0.0760	0.1804	154K	703K
emb-tfidf-256	0.0792	0.1829	171K	768K
noname-emb-tfidf	0.0893	0.1886	70K	329K
ing64-emb-tfidf	0.0823	0.1811	167K	754K
emb-tfidf-deep	0.1023	0.2103	301K	1.3M

На основі отриманих результатів можна зробити такі висновки:

– зважене вагування корисне для кодування з множинною активацією, зменшення втрат на 17%
– підходи на основі вкладень значно переважають кодування з множинною активацією: на 40% нижчі втрати при 13% меншому розмірі;
– для моделей з вбудованими представленнями різниця між усередненням та зваженим усередненням мінімальна, тому в цьому випадку середня агрегація є кращою через простоту реалізації;

– експерименти з розмірністю показали, що найкращий результат в нашому випадку дає розмір вкладення страви рівний 128. Несподіване погіршення втрат при розмірі 256 вказує на перенавчання через надмірну ємність моделі для набору даних з 3071 зразка;
– модель, що не враховує назви страв має наймешний розмір, але має вагомо меншу точність;
– збільшення розмірності представлення інгредієнтів погіршує втрати через можливе перенавчання;
– глибша мережа призводить до перенавчання на невеликому наборі даних та показує значне погіршення всіх метрик.

В результаті експериментів еталонною було визначено еталонну конфігурацію моделі – вкладення з усередненням, розмір вкладень для страв – 128. Ця конфігурація показує середньоквадратичну похибку 0.0768, абсолютну похибку 0.1798, розмір моделі – 703 кілобайти.

Після конвертації в TensorFlow Lite формат розмір моделі додатково зменшився до 323 кілобайт. Це вже прийнятний розмір для ресурсу мобільного застосунку. Квантизація динамічного діапазону додатково зменшила розмір моделі до 97 кілобайт, проте для застосунку було вирішено використовувати модель без квантизації, так як зменшення розміру на 200 кілобайт є не таким важливим, як зменшення точності, в даному випадку.

3.2 Тестування виведення висновків

Після успішного навчання моделі та її інтеграції в мобільний застосунок, було проведено експериментальне дослідження продуктивності виведення висновків безпосередньо на пристрої для платформ Android та iOS. Метою тестування було визначення швидкості обчислення вкладень та загальної придатності моделі для

використання в реальних умовах на мобільних пристроях користувачів на основі отриманого користувацького досвіду.

Для тестування на платформі Android використовувався смартфон Huawei Honor 200 з процесором Qualcomm Snapdragon 7 Gen 3. Для тестування на платформі iOS використовувався симулятор Xcode з iOS 17.5, конфігурація iPhone 15 Pro, симулятор використовує ресурси робочого комп'ютера MacBook з процесором M1 Pro.

Для оцінки продуктивності, окрім візуального сприйняття, було визначено 3 тестові сценарії:

– час ініціалізації моделі (ініціалізації інтерпритатора TensorFlow Lite);
– час виведення першого висновку (включає накладні витрати на виділення пам’яті та прогрівання процесора або прискорювача);
– середній час виведення висновку при 100 послідовних викликах.

Для точного вимірювання часу виведення було додано логування у платформо-специфічні реалізації. Замір результатів виконувався на debug збірках. Результати вимірювань швидкодії наведено в таблиці 3.2.

Таблиця 3.2 – Швидкодія виведення висновків на обох платформах

Тест	Android	iOS
Ініціалізація	24.9 мс	11.07 мс
Перший висновок	3.4 мс	3.05 мс
Середній висновок	0.28 мс	0.04 мс

Результати тестування виявили незначну різницю у продуктивності між платформами на користь iOS. Причинами такої різниці є те, що симулятор iOS безпосередньо використовує архітектуру Apple Silicon (M1 Pro) з уніфікованою пам'яттю [77], TensorFlow Lite на iOS оптимізований для роботи з графічним прискорювачем Metal [78].

Експериментальне тестування довело, що виведення на мобільних пристроях відбувається майже миттєво, навіть холодний старт відбувається за ~3 мілісекунди. Ці результати значно нижче порогу людського сприйняття (близько 100 мс для відчуття миттєвості відповіді) [29] та забезпечують плавний досвід користувача,

що було підтверджено особисто: користувач не відчуває затримок при взаємодії із застосунком, рекомендації генеруються "на льоту" без необхідності попереднього обчислення. Типова затримка запиту до хмарного сервера становить 50-200 мс (мережевий обмін) [7]; додатково потрібен час на серверне виведення. Виведення на пристрої до 1000 разів швидше ніж хмарний підхід для нашого сценарію (0,04 мс проти 50 мс).

У рамках даної роботи повноцінна оцінка якості рекомендацій за всіма категоріями методів не була проведена з об'єктивних причин:

– розроблена система є прототипом, що не має реальної бази користувачів, а отже, відсутні історичні дані взаємодій, необхідні для проведення офлайн-оцінки за метриками якості ранжування, різноманітності та покриття;
– проведення дослідження за участі користувачів потребує залучення репрезентативної групи учасників, розробки методології дослідження та статистичного аналізу результатів, що виходить за межі технічної спрямованості даної роботи;
– онлайн-експерименти можливі лише після публікації застосунку та набору критичної маси активних користувачів.

Натомість, у роботі проведено технічну валідацію системи, що включає: аналіз якості реконструкції автоенкодера через функції втрат на тренувальних даних; тестування коректності виведення висновків на Android та iOS; вимірювання продуктивності. Для повноцінної оцінки якості рекомендацій рекомендується провести пілотне тестування з групою реальних користувачів після публікації застосунку.

Висновки до розділу 3

У третьому розділі проведено експериментальне дослідження розробленої системи та підтверджено її ефективність. Реалізовано повний конвеєр навчання

моделі мовою Python з використанням бібліотеки TensorFlow на пристрої MacBook Pro з процесором M1 Pro. Проведено серію з 10 експериментів для оптимізації архітектури моделі. Встановлено оптимальні гіперпараметри: розмірність векторного представлення 128, розмірність представлення інгредієнтів 32, два приховані шари [256, 128], темп навчання 0,001, розмір пакету 128, 20 циклів навчання. Найкраща конфігурація досягла значення функції втрат 0,0784 та середньої абсолютної похибки 0,1824. Експериментально підтверджено, що кодування на основі векторних представлень на 40% ефективніше за кодування з множинною активацією, а використання назв страв покращує якість на 11%. Виконано тестування виведення висновків моделі на мобільних пристроях. На платформі Android отримано: час ініціалізації моделі 16,76 мс, холодний старт 34,14 мс, типовий час виведення 0,07 мс. На платформі iOS: час ініціалізації 11,07 мс, холодний старт 3,05 мс, типовий час виведення 0,04 мс. Виведення на пристрої значно швидше за хмарні рішення (0,04-0,07 мс проти 50-200 мс мережевого запиту). Це значно нижче порогу людського сприйняття і забезпечує приємний досвід користувача. Розмір моделі після квантизації становить менше 100 Кб. Результати підтверджують практичну цінність розробленого рішення та придатність для реального використання на мобільних пристроях.

ВИСНОВОК

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання розробки персоналізованої рекомендаційної системи для підбору страв у мобільному застосунку з використанням методів машинного навчання. На основі проведеного дослідження сформульовано наступні висновки:

1. Проведено аналіз існуючих підходів до побудови рекомендаційних систем для харчування, який показав, що, за умови виведення висновків на пристрої користувача, найбільш перспективним є підхід на основі вмісту з використанням глибокого навчання. Виявлено, що виведення безпосередньо на мобільному пристрої забезпечує переваги в швидкості, конфіденційності та автономній роботі порівняно з хмарними рішеннями.
2. Досліджено та реалізовано методи представлення характеристик страв для машинного навчання. Розроблено гібридний підхід до кодування, що поєднує: векторні представлення категоріальних ознак, текстову векторизацію назв страв та чисельні характеристики страви.
3. Розроблено методику холодного старту для швидкого формування початкового профілю користувача. Розроблено архітектуру нейронної мережі автокодувальника для формування векторних представлень страв. Навчання проводилось на ретельно очищеному та стандартизованому наборі даних з 3071 страви, було проведено уніфікацію інгредієнтів.
4. Розроблено мультиплатформенний мобільний застосунок на базі технології Kotlin Multiplatform для платформ Android та iOS, відсоток спільного коду понад 90%. Платформо-специфічний код обмежений інтеграцією з TensorFlow Lite та налаштуваннями системи.
5. Проведено серію з 10 експериментів для оптимізації архітектури моделі. Встановлено оптимальні гіперпараметри. Експериментальне тестування виведення на пристрої підтвердило високу продуктивність розробленого рішення.

Практичне значення отриманих результатів підтверджується створенням повнофункціонального мобільного застосунку, готового до розгортання. Розроблена система забезпечує: персоналізовані рекомендації страв на основі індивідуальних вподобань; повну автономність роботи без підключення до мережі; конфіденційність даних користувача; нульові операційні витрати на серверну інфраструктуру. Система може бути адаптована для ресторанного бізнесу, сервісів доставки їжі та дієтичного планування.

Напрями подальших досліджень: збільшення набору даних до 10+ тисяч страв для тестування глибших архітектур; впровадження механізмів уваги для агрегації інгредієнтів; використання попередньо навчених мовних моделей для кодування назв; реалізація багатозадачного навчання з додатковими виходами (класифікація типу кухні, прогнозування типу страви); розробка механізму динамічного оновлення моделі без перевстановлення застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Trattner C., Elsweiler D. Food recommender systems: important contributions, challenges and future research. 2017. 34 p.
2. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. 2nd ed. New York : Springer, 2015. 1016 p.
3. Min W., Jiang S., Liu L., Rui Y., Jain R. A survey on food computing. ACM Computing Surveys (CSUR). 2019. Vol. 52, No. 5. P. 1–36.
4. Burke R. Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction. 2002. Vol. 12, No. 4. P. 331–370.
5. Lops P., de Gemmis M., Semeraro G. Content-based recommender systems: State of the art and trends // Recommender Systems Handbook. Boston, MA : Springer, 2011. P. 73–105.
6. Su X., Khoshgoftaar T. M. A survey of collaborative filtering techniques. Advances in Artificial Intelligence. 2009. Vol. 2009. Article ID 421425. 19 p.
7. Burke R. Knowledge-based recommender systems. Encyclopedia of Library and Information Science. 2000. Vol. 69, No. 32. P. 180–200.
8. Pazzani M. J. A framework for collaborative, content-based and demographic filtering. Artificial Intelligence Review. 1999. Vol. 13, No. 5–6. P. 393–408.
9. Salton G., Buckley C. Term-weighting approaches in automatic text retrieval. Information Processing & Management. 1988. Vol. 24, No. 5. P. 513–523.
10. Mooney R. J., Roy L. Content-based book recommending using learning for text categorization // Proceedings of the Fifth ACM Conference on Digital Libraries. 2000.
    P. 195–204.
11. Pazzani M., Billsus D. Learning and revising user profiles: The identification of interesting web sites. Machine Learning. 1997. Vol. 27, No. 3. P. 313–331.
12. Rocchio J. J. Relevance feedback in information retrieval // The SMART Retrieval System: Experiments in Automatic Document Processing. Prentice-Hall, 1971. P. 313–323.

13. Batmaz Z., Yurekli A., Bilge A., Kaleli C. A review on deep learning for recommender systems: challenges and remedies. Artificial Intelligence Review. 2019. Vol. 52, No.
1. P. 1–37.
14. Bank D., Koenigstein N., Giryes R. Autoencoders // Machine Learning for Data Science Handbook. Springer, 2023. P. 353–374.
15. Hidasi B., Karatzoglou A., Baltrunas L., Tikk D. Session-based recommendations with recurrent neural networks // Proceedings of the 4th International Conference on Learning Representations (ICLR). 2016. 10 p.
16. Salvador A., Hynes N., Aytar Y., Marin J., Ofli F., Weber I., Torralba A. Learning cross- modal embeddings for cooking recipes and food images // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. P. 3020–3028.
17. He X., Liao L., Zhang H., Nie L., Hu X., Chua T. S. Neural collaborative filtering // Proceedings of the 26th International Conference on World Wide Web. 2017. P. 173–182.
18. Wu S., Tang Y., Zhu Y., Wang L., Xie X., Tan T. Session-based recommendation with graph neural networks // Proceedings of the AAAI Conference on Artificial Intelligence. 2019. Vol. 33, No. 01. P. 346–353.
19. Gomez-Uribe C. A., Hunt N. The netflix recommender system: Algorithms, business value, and innovation. ACM Transactions on Management Information Systems (TMIS). 2015. Vol. 6, No. 4. Article 13. 19 p.
20. Covington P., Adams J., Sargin E. Deep neural networks for YouTube recommendations // Proceedings of the 10th ACM Conference on Recommender Systems. 2016. P. 191–198.
21. Choi H., Lee H. A study on the recommendation algorithm for food recipes using taste information. Journal of the Korea Society of Computer and Information. 2017. Vol. 22, No. 1. P. 97–104.
22. Ge M., Ricci F., Massimo D. Health-aware food recommender system // Proceedings of the 9th ACM Conference on Recommender Systems. 2015. P. 333–334.
23. Recipes | KitchenAid. URL: https://www.kitchenaid.com/recipes.html (date of access: 15.08.2025).

24. Forbes H., Scholz D. U. Introducing the Recipe Recommendation API // Proceedings of the 1st Workshop on Recommender Systems and the Social Web. 2011. P. 1–4.
25. Freyne J., Berkovsky S. Intelligent food planning: personalized recipe recommendation // Proceedings of the 15th International Conference on Intelligent User Interfaces. 2010. P. 321–324.
26. Harvey M., Ludwig B., Elsweiler D. You are what you eat: learning user tastes for rating prediction // International Symposium on String Processing and Information Retrieval. Springer, 2013. P. 153–164.
27. Toledo R. Y., Alzahrani A. A., Martinez L. A food recommender system considering nutritional information and user preferences. IEEE Access. 2019. Vol. 7. P. 96695–96711.
28. Nguyen T. T., Hui P. M., Harper F. M., Terveen L., Konstan J. A. Exploring the filter bubble: the effect of using recommender systems on content diversity // Proceedings of the 23rd International Conference on World Wide Web. 2014. P. 677–686.
29. Sarwar B., Karypis G., Konstan J., Riedl J. Analysis of recommendation algorithms for e-commerce // Proceedings of the 2nd ACM Conference on Electronic Commerce. 2000. P. 158–167.
30. Knijnenburg B. P., Kobsa A., Jin H. Dimensionality of information disclosure behavior. International Journal of Human-Computer Studies. 2013. Vol. 71, No. 12. P. 1144–1162.
31. Lane N. D., Bhattacharya S., Georgiev P., Forlivesi C., Jiao L., Qendro L., Kawsar F. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices // Proceedings of the 15th International Conference on Information Processing in Sensor Networks. 2016. Article 23. 12 p.
32. Zhang Y., Chen X. Explainable recommendation: A survey and new perspectives. Foundations and Trends in Information Retrieval. 2020. Vol. 14, No. 1. P. 1–101.
33. TensorFlow Lite Guide. URL: https://www.tensorflow.org/lite/guide (date of access: 05.10.2025).
34. Ignatov A., Timofte R., Kulik A., Yang S., Wang K., Baum F., Wu M., Xu L., Van Gool
L. AI benchmark: All about deep learning on smartphones in 2019 // Proceedings of

the IEEE/CVF International Conference on Computer Vision Workshop. 2019. P. 3617–3635.

35. Krishnamoorthi R. Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342. 2018. 41 p.
36. PyTorch Mobile Documentation. URL: https://pytorch.org/mobile/home/ (date of access: 21.09.2025).
37. ONNX Runtime. URL: https://onnxruntime.ai/ (date of access: 21.09.2025).
38. Core ML Documentation. URL: https://developer.apple.com/documentation/coreml (date of access: 21.09.2025).
39. El-Kassas W. S., Abdullah B. A., Yousef A. H., Wahba A. M. Taxonomy of cross-platform mobile applications development approaches. Ain Shams Engineering Journal. 2017. Vol. 8, No. 2. P. 163–190.
40. Kotlin Multiplatform Documentation. URL: https://kotlinlang.org/docs/multiplatform.html (date of access: 09.11.2025).
41. React Native documentation. URL: https://reactnative.dev/ (date of access: 02.10.2025).
42. Flutter documentation. URL: https://flutter.dev/ (date of access: 02.10.2025).
43. .NET Multi-platform App UI (MAUI). URL: https://dotnet.microsoft.com/en-us/apps/maui (date of access: 02.10.2025).
44. Ionic Framework documentation. URL: https://ionicframework.com/docs (date of access: 02.10.2025).
45. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge : MIT Press, 2016. 800 p.
46. Liang D., Krishnan R. G., Hoffman M. D., Jebara T. Variational autoencoders for collaborative filtering // Proceedings of the 2018 World Wide Web Conference. 2018. P. 689–698.
47. Marin J., Biswas A., Ofli F., Hynes N., Salvador A., Aytar Y., Weber I., Torralba A. Recipe1M+: A dataset for learning cross-modal embeddings for cooking recipes and food images. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2021. Vol. 43, No. 1. P. 187–203.

48. Epirecipes Dataset. URL: https://www.kaggle.com/datasets/hugodarwood/epirecipes (date of access: 14.09.2025).
49. Food Ingredients and Recipe Dataset with Images. URL: https://www.kaggle.com/datasets/pes12017000148/food-ingredients-and-recipe-dataset-with-images (date of access: 14.09.2025).
50. Yummly-28K Recipes Dataset. URL: https://www.kaggle.com/datasets/mrsimple07/yummly-recipes-dataset (date of access: 14.09.2025).
51. Elsweiler D., Harvey M., Ludwig B., Said A. Bringing the "healthy" into food recommenders // Proceedings of the 7th Workshop on Recommender Systems and the Social Web. 2015. P. 33–36.
52. Balabanović M., Shoham Y. Fab: content-based, collaborative recommendation. Communications of the ACM. 1997. Vol. 40, No. 3. P. 66–72.
53. Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Devin M., Ghemawat S., Irving G., Isard M. et al. TensorFlow: A system for large-scale machine learning // 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 2016. P. 265–283.
54. Grbovic M., Cheng H. Real-time personalization using embeddings for search ranking at Airbnb // Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018. P. 311–320.
55. Majumder B. P., Li S., Ni J., McAuley J. Generating personalized recipes from historical user preferences // Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. 2019. P. 5976–5982.
56. Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research. 2014. Vol. 15, No. 1. P. 1929–1958.
57. Kingma D. P., Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. 2014.
58. Prechelt L. Early stopping-but when? // Neural Networks: Tricks of the trade. Springer, 1998. P. 55–69.

59. Schein A. I., Popescul A., Ungar L. H., Pennock D. M. Methods and metrics for cold- start recommendations // Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval. 2002. P. 253–260.
60. Ziegler C. N., McNee S. M., Konstan J. A., Lausen G. Improving recommendation lists through topic diversification // Proceedings of the 14th international conference on World Wide Web. 2005. P. 22–32.
61. Kotlin Multiplatform Wizard. URL: https://terrakok.github.io/kmp-web-wizard/ (date of access: 20.09.2025).
62. Decompose documentation. URL: https://arkivanov.github.io/Decompose/ (date of access: 21.09.2025).
63. Koin documentation. URL: https://insert-koin.io/ (date of access: 21.09.2025).
64. Room Kotlin Multiplatform. URL: https://developer.android.com/kotlin/multiplatform/room (date of access: 21.09.2025).
65. Kotlinx Serialization documentation. URL: https://github.com/Kotlin/kotlinx.serialization (date of access: 21.09.2025).
66. Sketch image loading library. URL: https://github.com/panpf/sketch (date of access: 21.09.2025).
67. Arrow functional programming library. URL: https://arrow-kt.io/ (date of access: 22.09.2025).
68. Kermit logging library. URL: https://github.com/touchlab/Kermit (date of access: 22.09.2025).
69. Kotlinx Coroutines documentation. URL: https://kotlinlang.org/docs/coroutines-overview.html (date of access: 22.09.2025).
70. Kotlinx Datetime library. URL: https://github.com/Kotlin/kotlinx-datetime (date of access: 22.09.2025).
71. Compottie library. URL: https://github.com/alexzhirkevich/compottie (date of access: 23.09.2025).
72. WebView Multiplatform library. URL: https://github.com/KevinnZou/compose-webview-multiplatform (date of access: 23.09.2025).

73. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.
74. Unidirectional Data Flow architecture. URL: https://developer.android.com/topic/architecture/ui-layer (date of access: 01.10.2025).
75. SQLite documentation. URL: https://www.sqlite.org/docs.html (date of access: 01.10.2025).
76. Database normalization and design. URL: https://docs.oracle.com/en/database/oracle/oracle-database/19/admin/managing-tables.html (date of access: 07.10.2025).
77. Apple Neural Engine overview. URL: https://machinelearning.apple.com/research/neural-engine (date of access: 11.10.2025).
78. TensorFlow Metal Plugin documentation. URL: https://developer.apple.com/metal/tensorflow-plugin/ (date of access: 13.10.2025).
79. Free Recipe API. URL: https://www.themealdb.com/api.php (date of access: 02.09.2025)
80. Redish repository. URL: https://github.com/as-i-see/Redish (date of access: 27.11.2025)
81. Алексіна, Л. Т., & Бондарчук, А. П. (2024). Оптимізація гіперпараметрів для машинного навчання. *Зв'язок*, (2), 18-22.
82. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., Hlushak, O., Zhyltsov, O., Ilich, L., Kobets, N., Kovaliuk, T., Kuchakovska, H., Lytvyn, O., Lytvyn, P., Mashkina, I., Morze, N., Nosenko, T., Proshkin, V., Radchenko, S., … Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science. https://doi.org/10.28925/9720213284km.