Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ ДЛЯ АНАЛІЗУ ТА СКЛАДАННЯ КУБИКА РУБІКА ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ КОМП'ЮТЕРНОГО ЗОРУ ТА ДОПОВНЕНОЇ РЕАЛЬНОСТІ

DEVELOPMENT OF A SOFTWARE SUITE FOR ANALYZING AND SOLVING RUBIK'S CUBE USING COMPUTER VISION AND AUGMENTED REALITY TECHNOLOGIES

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Косарєв Владислав Олегович

Керівник:
Потейчук Михайло Іванович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Косарєву Владиславу Олеговичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка програмного комплексу для аналізу та складання кубика Рубіка за допомогою технологій комп'ютерного зору та доповненої реальності»*
Керівник роботи: *асистент, Потейчук М. І.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ, вимоги до розробки програмного забезпечення, ергономічні вимоги до функціонування програмного засобу.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз і огляд сучасного стану проблеми автоматизації складання кубика Рубіка та формулювання актуальних задач дослідження, обґрунтувати вибір апаратних і програмних засобів, алгоритмів розпізнавання й вирішення головоломки, а також спроєктувати програмне забезпечення із застосуванням технологій комп'ютерного зору та доповненої реальності, інтегрувати ключові модулі, проєктування бази даних для збереження результатів і аналітики, а також тестування та налагодження комплексу, розглянути заходи інформаційної безпеки, захисту персональних даних користувачів і визначити перспективи подальшого вдосконалення системи.*
5. Перелік графічного матеріалу: *17 рисунків*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Потейчук М. І.*		
Специфікація вимог до розробленої системи	*Потейчук М. І.*		
Розробка об'єкта проектування	*Потейчук М. І.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	979 %		
Академічна доброчесність	*Потейчук М. І.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Виконано
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	Виконано
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	Виконано
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	Виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Виконано

Здобувач вищої освіти Владислав КОСАРЄВ

Керівник кваліфікаційної роботи Михайло ПОТЕЙЧУК

АНОТАЦІЯ

Косарєв В. О. Розробка програмного комплексу для аналізу та складання кубика Рубіка за допомогою технологій комп’ютерного зору та доповненої реальності. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається з вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі виконано аналіз сучасного стану технологій розпізнавання й автоматизації складання кубика Рубіка, розглянуто основні програмні рішення, алгоритми обробки зображень і підходи до застосування доповненої реальності. Сформульовано завдання дослідження та окреслено технічні вимоги до майбутньої системи.

У другому розділі проаналізовано функціональні й технічні вимоги до програмного комплексу; спроєктовано програмну архітектуру, розроблено UML-діаграми взаємодії модулів, обґрунтовано вибір апаратної платформи та алгоритмів розпізнавання кольорів і вирішення головоломки. Визначено структуру бази даних для збереження результатів та аналітики.

У третьому розділі описано практичну реалізацію комплексу: створення програмних модулів комп’ютерного зору, інтеграцію AR-підказок, тестування й налагодження роботи на мобільних пристроях, впровадження заходів інформаційної безпеки та проведення експериментального аналізу ефективності системи. Наведено результати експериментів та рекомендації щодо перспектив розвитку.

Ключові слова: кубик Рубіка, комп’ютерний зір, мобільний додаток, доповнена реальність, алгоритми вирішення, база даних.

ABSTRACT

Kosarev V. Development of a software suite for analyzing and solving rubik's cube using computer vision and augmented reality technologies. Manuscript. Qualification work of the bachelor's degree program "Software Engineering" specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification work consists of an introduction, three sections, conclusions, and a list of sources used.

The first section analyzes the current state of technologies for recognizing and automating the assembly of a Rubik's cube, considers the main software solutions, image processing algorithms, and approaches to the application of augmented reality. The research tasks are formulated and the technical requirements for the future system are outlined.

The second section analyzes the functional and technical requirements for the software complex; the software architecture is designed, UML diagrams of module interaction are developed, and the choice of the hardware platform and algorithms for color recognition and puzzle solving are justified. The database structure for storing results and analytics is defined.

The third section describes the practical implementation of the complex: creation of computer vision software modules, integration of AR prompts, testing and debugging on mobile devices, implementation of information security measures and experimental analysis of the system's effectiveness. The results of the experiments and recommendations for development prospects are presented.

Keywords: Rubik's cube, computer vision, mobile application, augmented reality, solution algorithms, database.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі спостерігається стрімкий розвиток інформаційних технологій, особливо у сфері комп'ютерного зору та доповненої реальності. Задачі розпізнавання об'єктів та автоматизації процесів вирішення складних головоломок стають все більш затребуваними як у науковій, так і в освітній та розважальній сферах. Кубик Рубіка виступає не лише класичним прикладом алгоритмічної задачі, але й зручним полігоном для апробації сучасних підходів до комп'ютерного зору, машинного навчання та візуальної навігації. Розробка програмного комплексу, який здатний аналізувати та розв'язувати кубик Рубіка із використанням мобільних пристроїв, відкриває нові можливості для навчання, тренування мислення та розвитку інженерних навичок. Додатково, інтеграція доповненої реальності дозволяє вивести взаємодію користувача із системою на якісно новий рівень, забезпечуючи інтуїтивність та ефективність процесу навчання. Це особливо актуально в умовах широкої доступності смартфонів та планшетів з потужними камерами і графічними процесорами.

Водночас, впровадження таких технологій дозволяє вирішувати низку актуальних задач: підвищення ефективності навчання, популяризація STEM-дисциплін, розвиток ринку мобільних освітніх застосунків, а також автоматизація рутинних процесів у практиці викладання й тренувань. Дослідження й удосконалення алгоритмів розпізнавання стану кубика та оптимального складання мають наукову новизну і практичну цінність для подальшого впровадження в інших сферах – робототехніці, інтелектуальних іграх, адаптивних навчальних системах. Таким чином, тема роботи є надзвичайно актуальною, відповідає світовим тенденціям розвитку IT-галузі та має вагоме практичне значення.

Метою даної кваліфікаційної роботи бакалавра є розробка програмного комплексу для автоматизованого аналізу та складання кубика Рубіка з використанням технологій комп'ютерного зору та доповненої реальності на

платформі мобільних пристроїв, а також дослідження ефективності застосування сучасних алгоритмів розпізнавання та вирішення даної головоломки.

Об’єктом дослідження є процеси аналізу та автоматизованого вирішення головоломки типу «кубик Рубіка» з використанням сучасних програмно-апаратних засобів.

Предметом дослідження є програмні алгоритми, методи комп’ютерного зору, принципи побудови модульних програмних комплексів та технології доповненої реальності для мобільних платформ, які застосовуються для аналізу та складання кубика Рубіка.

Виходячи з мети роботи, було сформульовано наступні завдання:

– провести огляд сучасних технологій, алгоритмів та програмних рішень для аналізу й складання кубика Рубіка;

– сформулювати вимоги до програмного комплексу, розробити функціональну та структурну архітектуру системи;

– реалізувати модулі комп’ютерного зору для розпізнавання стану кубика на відеопотоці з камери;

– інтегрувати алгоритми автоматизованого вирішення головоломки з мінімальною кількістю ходів;

– розробити модуль доповненої реальності для надання інтерактивних покрокових підказок користувачу;

– спроектувати й реалізувати базу даних для збереження результатів, статистики та аналітики;

– провести тестування програмно-апаратного комплексу, оцінити якість, продуктивність та перспективи подальшого розвитку системи.

РОЗДІЛ 1
АНАЛІЗ ТЕХНОЛОГІЙ ТА ПРОГРАМНИХ КОМПЛЕКСІВ ДЛЯ АНАЛІЗУ ТА СКЛАДАННЯ КУБИКА РУБІКА І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Автоматизація процесу аналізу та складання кубика Рубіка у XXI столітті стала не просто захопленням ентузіастів чи сферою дитячої освітньої гри, а предметом ґрунтовних досліджень у сучасній науці, де поєднуються здобутки інформатики, математичного моделювання, когнітивних технологій, комп’ютерного зору та робототехніки. Зміни у підходах до вирішення цієї класичної головоломки наочно ілюструють динаміку прогресу в ІТ-галузі, а також демонструють реальне застосування наукових теорій у практиці проєктування інтелектуальних систем.

Поява кубика Рубіка у 1974 році стала поштовхом для генерації величезної кількості комбінаційних теорій та пошукових стратегій, що протягом перших десятиліть обмежувалися переважно «паперовими» алгоритмами й популяризацією ручного спідкубінгу. Проте вже у 1980-х роках із появою персональних комп’ютерів були зроблені перші спроби алгоритмічного аналізу: програмісти створювали симулятори, які імітували механіку кубика та дозволяли експериментувати з варіантами складання. На рисунку 1.1 представлено систему комп’ютерного зору, адаптивне навчання та AR-технології.

З кінця 1990-х і початку 2000-х років, із розвитком мов програмування високого рівня та появою доступних засобів візуалізації, програмні комплекси для аналізу кубика Рубіка набули дедалі більшої універсальності. Перші серйозні програми вже не просто імітували окремі комбінації, а й могли автоматично знаходити шлях від довільної конфігурації до цільового стану. Тут почала активно застосовуватись теорія груп, комбінаторний аналіз та

алгоритми пошуку найкоротшого шляху у графах, що підготувало ґрунт для подальшої алгоритмічної революції.

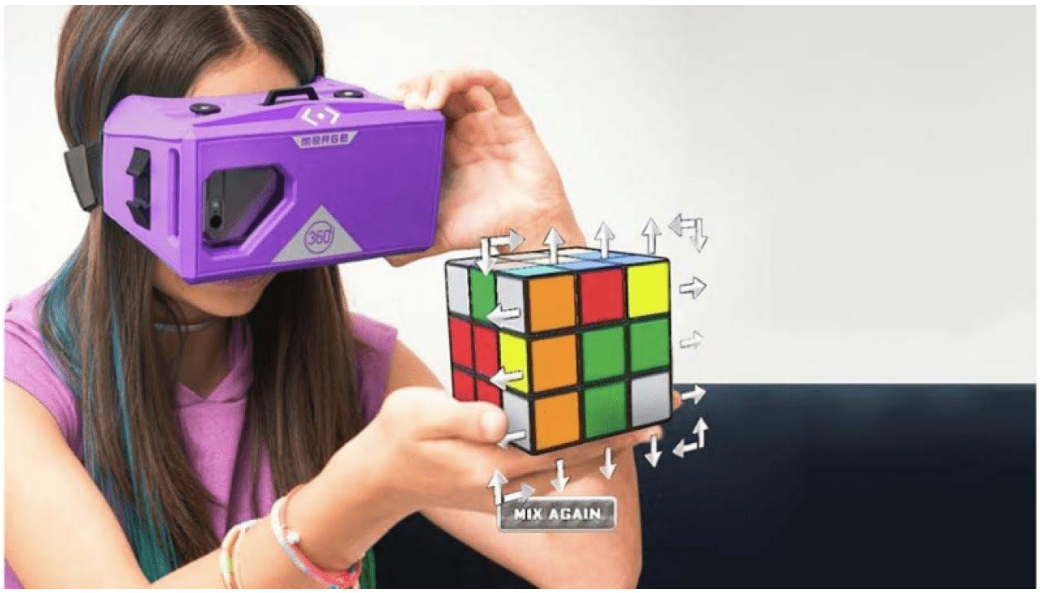

Рисунок 1.1 – Еволюція програмних рішень для автоматизації складання кубика Рубіка [1]

З кінця 1990-х і початку 2000-х років, із розвитком мов програмування високого рівня та появою доступних засобів візуалізації, програмні комплекси для аналізу кубика Рубіка набули дедалі більшої універсальності. Перші серйозні програми вже не просто імітували окремі комбінації, а й могли автоматично знаходити шлях від довільної конфігурації до цільового стану. Тут почала активно застосовуватись теорія груп, комбінаторний аналіз та алгоритми пошуку найкоротшого шляху у графах, що підготувало ґрунт для подальшої алгоритмічної революції.

На початку 2010-х років відбулося принципове переосмислення підходів до задачі: якщо раніше стан кубика вводили вручну або зчитували з простого графічного редактора, то сучасний світ вимагав повної автоматизації. Справжній прорив відбувся завдяки технологіям комп’ютерного зору – автоматичного розпізнавання вхідних даних із реального світу за допомогою цифрових камер, глибокого навчання, класифікації й обробки зображень.

Сучасні конволюційні нейронні мережі (CNN) дозволили ідентифікувати кольори граней із точністю понад 98 % навіть у складних умовах освітлення, а також автоматично «відрізняти» зношені, затерті або нестандартні наклейки від основних кольорів (рис. 1.2).

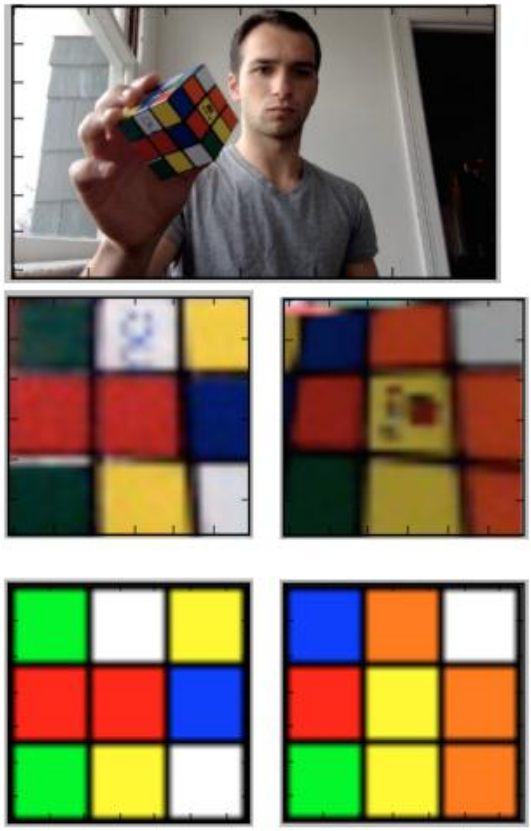

Рисунок 1.2 – Система комп’ютерного зору на базі глибокої згорткової нейронної мережі (CNN) для автоматичного аналізу фізичного стану кубика Рубіка, зчитування кольорів, сегментація зображення та формування цифрової моделі [2]

У практиці сучасного програмування найбільш популярними бібліотеками для розпізнавання стану кубика є OpenCV (2024), TensorFlow, PyTorch, а також спеціалізовані Python-модулі, призначені для класифікації кольорів та обробки відеопотоків у режимі реального часу. Сучасні мобільні додатки на їх основі здатні за лічені секунди отримати дані про положення

кожного елементу головоломки та автоматично перейти до наступного етапу – розрахунку оптимального розв’язку [2].

Після розпізнавання поточного стану головоломки постає фундаментальне завдання – знайти оптимальну або близьку до оптимальної послідовність дій для її складання. Тут алгоритмічна еволюція пройшла шлях від класичного brute force до комбінованих, двофазних, евристичних і AI-підходів.

Ключовим етапом став двофазний алгоритм Kociemba, що сьогодні є стандартом де-факто для програмних комплексів у цій сфері. Суть алгоритму – розділення пошуку на дві підзадачі, кожна з яких значно скорочує простір рішень, а загальна кількість ходів рідко перевищує 20–22 (майже математична межа для кубика 3x3x3). Блок-схему його роботи, де послідовно показано проходження фаз та принципи оптимізації вибору ходів, зображено на рисунку 1.3.

Опис блок-схеми двофазного алгоритму Кочімби, що демонструє структурну організацію етапів:

– вхідний стан кубика – отримання колірної інформації за допомогою комп’ютерного зору (наприклад, OpenCV);

– цикл зчитування 6 граней – кожен кадр аналізується й записується у масив стану;

– перший етап (Phase 1) – пошук послідовності ходів, яка переводить кубик у підгрупу G_1 (керована тільки двохоборотними поворотами R2, L2, F2, B2) із метою нормалізації орієнтацій граней і країв;

– збереження масиву даних – підготовка стану до другої фази;

– другий етап (Phase 2) – обчислення рухів, щоб вирішити кубик повністю, починаючи з G_1, з урахуванням евристичних таблиць (pattern databases);

– відображення інструкцій (іноді з візуалізацією в 3D) – кроки вирішення транслюються в інтерфейс (CLI, GUI або AR-доповнення).

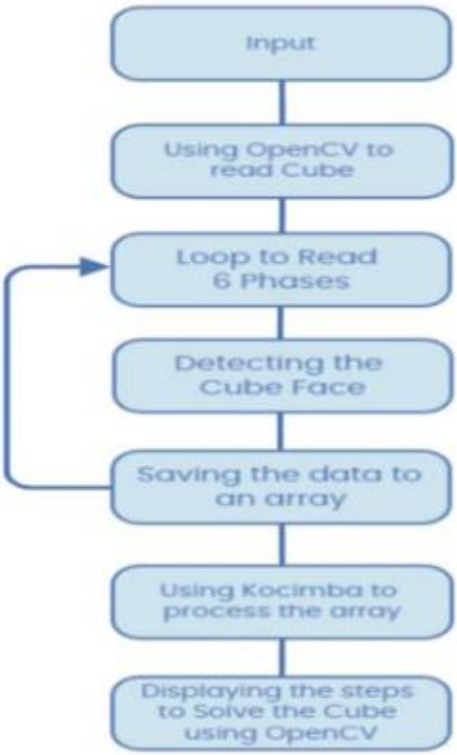

Рисунок 1.3 – Блок-схема двофазного алгоритму Kociemba для вирішення кубика Рубіка [3]

Завдяки відкритим Python-бібліотекам (наприклад, kociemba-python), сучасні мобільні додатки можуть виконувати розрахунок у реальному часі навіть на недорогих пристроях. Але у 2020-х роках стали популярними й більш просунуті підходи. Це – гібридні AI-системи, які поєднують класичні алгоритми (Kociemba, IDA*) з модулями глибокого навчання, reinforcement learning та евристичним аналізом патернів (рис. 1.4). Гібридна архітектура дозволяє програмі самостійно навчатися на основі мільйонів розв’язків, адаптуватися до нових типів головоломок і навіть «генерувати» унікальні шляхи для нестандартних комбінацій, які не зустрічалися у навчальному наборі [3].

Іще одним проривом стали генетичні й еволюційні алгоритми, що, на відміну від класичних пошукових методів, використовують ідеї природного добору, мутацій і кросоверів для пошуку «найкращих» шляхів до цілі у складних або нетипових ситуаціях. Такі системи знаходять застосування там,

де початковий стан головоломки дуже віддалений від класичних, або коли необхідна адаптація під особливі фізичні обмеження маніпулятора. Дослідження демонструють, що гібридні алгоритми часто забезпечують більшу гнучкість та універсальність у порівнянні з вузько оптимізованими класичними підходами.

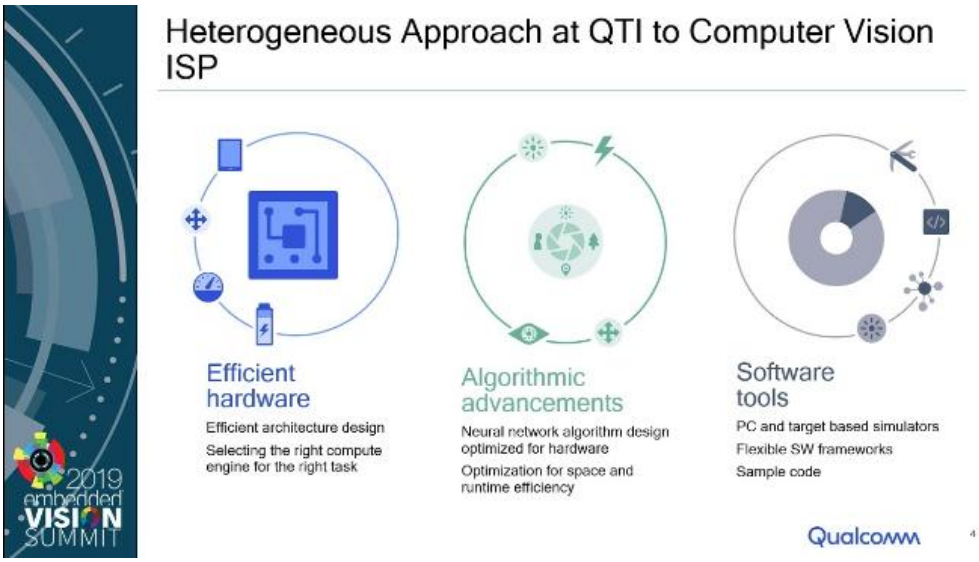

Рисунок 1.4 – Архітектура сучасної гібридної системи аналізу та складання [4]

Паралельно з алгоритмічним розвитком, у 2020-2025 рр. відбувається справжня революція у сфері візуалізації й взаємодії з користувачем. Найбільш перспективним напрямом є впровадження доповненої реальності (AR), яка забезпечує візуальний супровід усіх етапів складання кубика. Мобільні застосунки на базі ARCore та ARKit накладають підказки, стрілки, маркери чи вказівки просто на екран смартфона або планшета, допомагаючи користувачу інтуїтивно правильно виконувати кожен хід (рис. 1.5).

Актуальні наукові дослідження підтверджують ефективність AR у навчальному контексті: інтеграція такої візуалізації у навчальні платформи скорочує час засвоєння типових комбінацій, знижує кількість помилок і підвищує мотивацію навіть у початківців. Особливої популярності набули додатки з гейміфікованими елементами: рейтингами, досягненнями,

змаганнями, збереженням історії складання та спільною статистикою для груп учнів чи колективів.

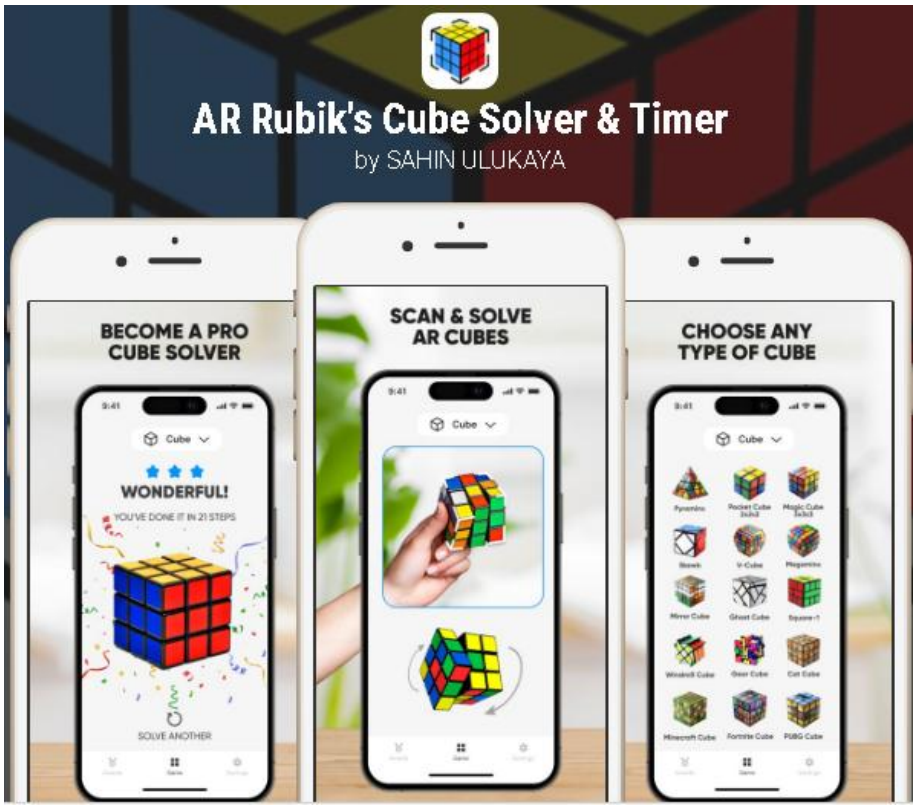

Рисунок 1.5 – Віртуальні AR-підказки в інтерфейсі мобільного застосунку для складання кубика Рубіка [5]

Ще одним напрямом, що бурхливо розвивається останніми роками, є створення робототехнічних систем для автоматичного фізичного складання кубика Рубіка. Якщо у 2000-х роках це були, переважно, експериментальні пристрої ентузіастів, то з розвитком Arduino, Raspberry Pi, дешевих сервоприводів і якісних веб-камер робототехнічні комплекси стали доступними для освітніх закладів, лабораторій і навіть шкіл. На рисунку 1.6 представлено структурну схему такого комплексу, камера високої роздільної здатності, модуль машинного зору, алгоритм пошуку рішення, блок прийняття рішень і механічний маніпулятор, який здійснює складання [6].

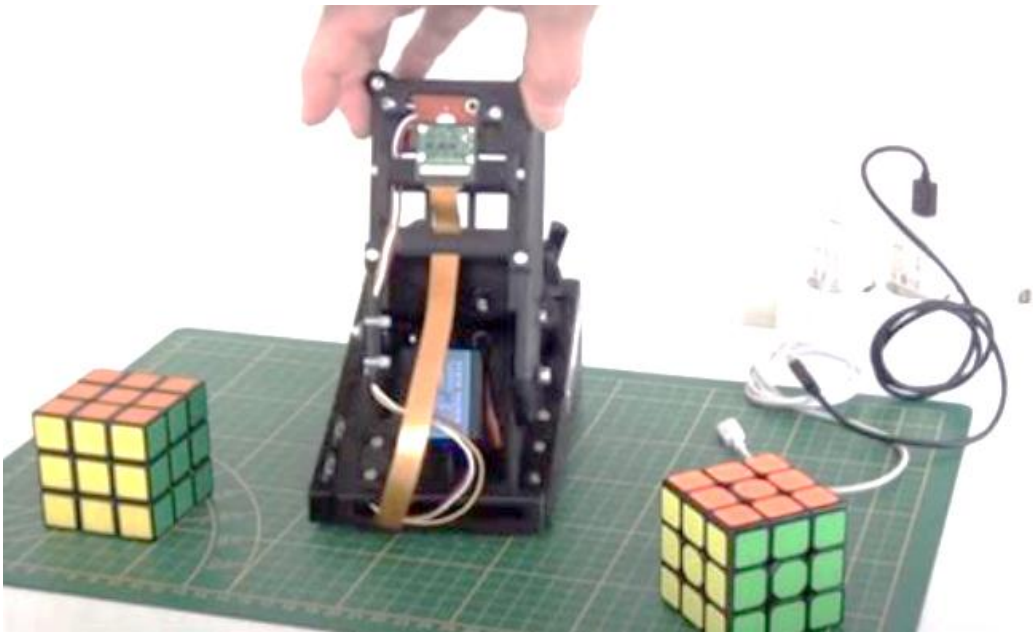

Рисунок 1.6 – Робототехнічна система автоматичного складання кубика Рубіка [6]

Сьогодні такі системи не лише демонструють унікальні швидкості складання (до 0,3-0,5 секунди), а й виступають платформою для практичного навчання програмуванню, електроніці, мехатроніці та оптимізаційним методам у рамках STEM-освіти. Наукова новизна подібних розробок полягає в синтезі всіх описаних раніше технологій: комп'ютерного зору, адаптивних алгоритмів, гнучких механічних маніпуляторів та інтерактивних інтерфейсів [6].

Попри значний прогрес, у розв'язанні задачі автоматизації складання кубика Рубіка зберігається ряд наукових і практичних викликів. Це – проблема універсального розпізнавання кольорів на «нестандартних» головоломках (наприклад, прозорих, металізованих, з нестандартними відтінками), оптимізація швидкодії на пристроях із низькою продуктивністю, інтеграція голосового керування та багатомовної підтримки, а також забезпечення розширеної підтримки для головоломок іншого типу (4x4x4, 5x5x5, Megaminx, Pyraminx тощо).

Наукові дослідження останніх років вказують на перспективи подальшої автоматизації завдяки застосуванню мультиагентних систем, розподілених

обчислень, хмарних технологій для збереження і аналізу статистики, а також глибшої гейміфікації процесу. Розвиток хмарних платформ і цифрових екосистем у поєднанні з AR/VR відкриває нові горизонти як для індивідуального навчання, так і для колективних тренінгів і наукових експериментів.

Загалом автоматизація аналізу та складання кубика Рубіка у сучасному світі – це не лише цікава прикладна задача, а й наочний полігон для апробації фундаментальних ідей сучасної математики, інформатики та інженерії, а також ефективний засіб для інтеграції науки у практику через доступні, цікаві й мотивуючі технологічні рішення.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Враховуючи сформульовану мету дослідження, а також специфічні технічні вимоги до архітектури та функціоналу розроблюваного програмного комплексу, було детально проаналізовано структуру проєкту та визначено низку пріоритетних завдань, виконання яких є необхідною умовою для досягнення запланованих результатів. Комплексний підхід до планування дозволив систематизувати етапи розробки, забезпечити логічну послідовність впровадження окремих модулів і гарантувати узгодженість усіх компонентів системи між собою. Ретельне формулювання цих завдань спрямоване на оптимізацію процесу створення програмно-апаратного комплексу, підвищення його ефективності, масштабованості та зручності для кінцевого користувача. Відповідно, розроблена стратегія реалізації проєкту передбачає вирішення цілого ряду ключових задач:

– провести огляд сучасних технологій, алгоритмів та програмних рішень для аналізу й складання кубика Рубіка;

– сформулювати вимоги до програмного комплексу, розробити функціональну та структурну архітектуру системи;

– реалізувати модулі комп’ютерного зору для розпізнавання стану кубика на відеопотоці з камери;

– інтегрувати алгоритми автоматизованого вирішення головоломки з мінімальною кількістю ходів;

– розробити модуль доповненої реальності для надання інтерактивних покрокових підказок користувачу;

– спроектувати й реалізувати базу даних для збереження результатів, статистики та аналітики;

– провести тестування програмно-апаратного комплексу, оцінити якість, продуктивність та перспективи подальшого розвитку системи.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Створення інтелектуальної системи для автоматизованого аналізу та складання кубика Рубіка із застосуванням комп'ютерного зору та доповненої реальності є складним міждисциплінарним завданням, що поєднує фундаментальні основи програмної інженерії, математичного моделювання, когнітивних технологій, штучного інтелекту й сучасних підходів до проєктування інтерактивних інформаційних систем. Аналіз і визначення вимог до такого програмного забезпечення базуються на вивченні цільових сценаріїв використання, особливостей користувацької взаємодії, сучасного стану технологій, а також досвіду провідних рішень галузі [7].

В першу чергу необхідно усвідомити, що основними користувачами розроблюваного комплексу виступають не лише ентузіасти головоломок чи професійні спідкубери, а й широка аудиторія – від школярів, студентів, педагогів до науковців і організаторів освітніх проєктів. Саме це обумовлює необхідність орієнтації системи на різні рівні складності й сценарії роботи: від інтерактивного навчання та самостійного тренування до змагальних режимів та аналітичних досліджень ефективності алгоритмів.

Науковий підхід до формування вимог передбачає комплексний аналіз як функціональних, так і нефункціональних параметрів. Функціональні вимоги містять автоматичне розпізнавання стану кубика на основі обробки зображень у реальному часі, підтримку стандартних і розширених типів головоломок, реалізацію ефективних алгоритмів пошуку рішення (Kociemba, IDA*, гібридних AI-підходів), а також інтеграцію AR-технологій для візуального супроводу та підказок користувачеві під час складання. Не менш важливою є можливість ведення персональної статистики, рейтингу, підтримки багатомовності,

підключення до смарт-кубиків чи робототехнічних маніпуляторів, а також експорт і синхронізація даних із хмарними платформами.

До нефункціональних вимог належать кросплатформенність (Android, iOS, Windows, MacOS, Web), висока швидкодія (реакція системи у реальному часі), точність розпізнавання (не менше 98 % у стандартних умовах), захист персональних даних, масштабованість, гнучка архітектура для подальшого розширення функціоналу, а також можливість автоматизованого тестування й оперативного оновлення компонентів [8].

Особливістю розроблюваної системи є її модульна архітектура, яка забезпечує ізоляцію функціональних блоків і полегшує масштабування та супровід ПЗ. Система поділяється на окремі компоненти: модуль обробки зображень, модуль комп'ютерного зору й класифікації, модуль генерації рішень, AR-модуль, модуль користувацького інтерфейсу, модуль аналітики й збереження, а також модуль інтеграції з апаратними пристроями (рис. 2.2). Такий підхід дає змогу легко додавати нові алгоритми чи розширювати підтримку пристроїв без зміни основної логіки роботи системи.

Модуль обробки зображень реалізує функції отримання відео з камери, попередньої фільтрації, вирівнювання кольору й сегментації. Модуль комп'ютерного зору здійснює глибоку класифікацію за допомогою CNN (наприклад, TensorFlow Lite), формує цифрову модель стану кубика й передає дані до генератора рішень. Ядро генератора містить реалізацію класичних та AI-алгоритмів (Kociemba, IDA*, reinforcement learning), вибір оптимального шляху залежно від конкретної конфігурації та вимог користувача. AR-модуль здійснює накладання 3D-підказок, стрілок, анімацій на відео (з підтримкою ARCore, ARKit), а модуль UI забезпечує доступність та зручність інтерфейсу, персоналізацію рівня складності, налаштування підказок та інші опції. Модуль аналітики відповідає за збереження результатів, ведення статистики, підтримку рейтингів, хмарну синхронізацію, гейміфікацію та аналітичну візуалізацію. Інтеграція із смарт-кубиками й робототехнічними маніпуляторами відбувається

через окремий модуль апаратного з’єднання (Bluetooth/Wi-Fi/USB), який може масштабуватись для підтримки нових пристроїв [8].

Всі модулі взаємодіють через чітко визначені API та подієві шини, що уможливлює асинхронну роботу, паралельне тестування та оперативне оновлення окремих компонентів.

У розроблюваному програмному комплексі передбачено різноманітні сценарії використання, які охоплюють освітні, змагальні, тренувальні й дослідницькі потреби. У навчальному режимі користувач сканує кубик через мобільний пристрій, отримує покрокові AR-підказки, вивчає алгоритмічні патерни та фіксує свій прогрес. У режимі спідкубінгу система забезпечує максимально швидке сканування й розрахунок оптимального рішення, відображає мінімальні підказки й фіксує досягнення користувача в персональному чи груповому рейтингу. У змагальному режимі система дозволяє організовувати колективні тренінги, онлайн-турніри, відслідковувати командні та індивідуальні результати, підтримує експорт аналітики для подальшого аналізу. У лабораторному/дослідницькому режимі система може використовуватись для тестування нових алгоритмів вирішення, підключення зовнішніх пристроїв, оцінки ефективності різних моделей розпізнавання чи генерації рішень, а також для апробації робототехнічних систем автоматичного складання кубика.

Використання UML-діаграм дозволяє не лише формально задокументувати основні сценарії, компоненти й взаємодії у системі, а й створити підґрунтя для командної роботи над проєктом, тестування, масштабування та подальшої еволюції комплексу. У даній роботі використано три ключові UML-діаграми, які відповідають загальноприйнятим підходам сучасної програмної інженерії [9].

Діаграма варіантів використання (Use Case Diagram) – рисунок 2.1 ілюструє основні ролі користувачів і зовнішніх пристроїв у системі, а також ключові сценарії взаємодії. Користувач (актор) може сканувати кубик, запускати AR-підказки, зберігати результати, переглядати аналітику,

підключатися до смарт-кубика або брати участь у групових сесіях. Смарт-кубик або робототехнічний маніпулятор також можуть ініціювати або приймати команди, а хмарний сервер відповідає за збереження рейтингів, історії складань і синхронізацію (рисунок виконано у стилі, адаптовано до даної задачі).

Рисунок 2.1 – UML-діаграма варіантів використання для системи аналізу та складання кубика Рубіка

Діаграма компонентів (Component Diagram) – рисунок 2.2, деталізує основну структурну організацію системи: показує, з яких модулів складається програмний комплекс, як саме вони взаємодіють через внутрішні інтерфейси (API), які основні потоки даних між ними, та які залежності існують. Кожний компонент чітко відділений На діаграмі простежується, як дані рухаються від камери через аналіз, генерацію рішення, AR-візуалізацію та збереження результатів, а також показані можливості розширення архітектури додаванням нових модулів:

Діаграма послідовності (Sequence Diagram) – рисунок 2.3, моделює типовий сценарій взаємодії між користувачем, ключовими модулями та пристроями системи у процесі аналізу та складання кубика. Вона демонструє ланцюжок викликів: користувач ініціює сканування кубика через інтерфейс,

модуль обробки зображень отримує відео, передає візуальні дані до модуля класифікації, той формує цифрову модель стану й надсилає у генератор рішень, який повертає оптимальний шлях складання, AR-модуль візуалізує покрокові підказки, після завершення результат фіксується у модулі аналітики та зберігається на сервері (усе виконано у стилі, із зазначенням учасників та потоків подій) [10].

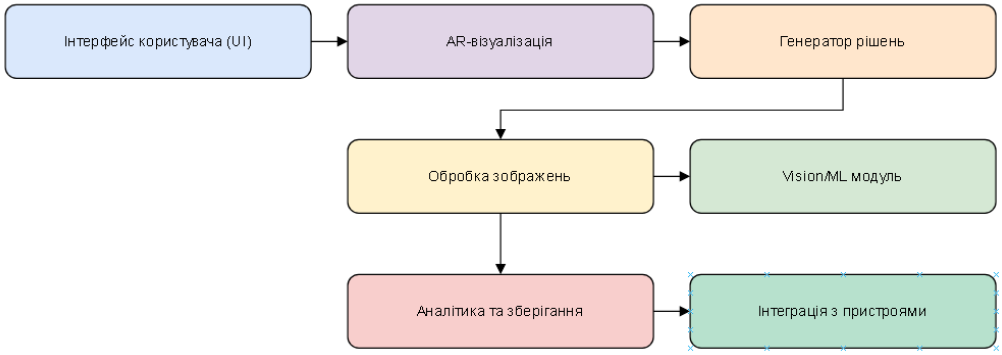

Рисунок 2.2 – UML-діаграма компонентів для програмного комплексу розпізнавання і складання кубика Рубіка

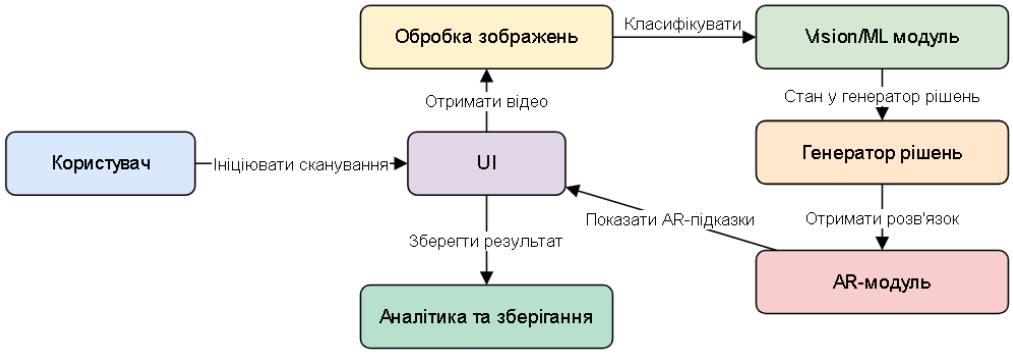

Рисунок 2.3 – UML-діаграма послідовності для сценарію «Сканування кубика та AR-візуалізація рішення»

Узагальнення вимог до програмного комплексу для аналізу та складання кубика Рубіка із застосуванням комп’ютерного зору та доповненої реальності підтверджує, що найбільш ефективним підходом є побудова відкритої, модульної архітектури з опорою на сучасні технології ML, AR, аналітики та

IoT. Формалізація архітектури через UML-діаграми, сценарії використання та структурні схеми дозволяє забезпечити гнучкість, масштабованість і подальшу еволюцію програмного забезпечення відповідно до сучасних трендів та реальних потреб користувача.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

У процесі розробки програмного комплексу для автоматизованого аналізу та складання кубика Рубіка із застосуванням технологій комп’ютерного зору та доповненої реальності на платформі Android ключовим завданням є виважений і науково обґрунтований вибір засобів, методів і алгоритмів для досягнення максимальної точності, швидкодії та користувацької зручності. Сучасний стан розвитку комп’ютерного зору та мобільного машинного навчання, зокрема на базі Android, відкриває широкі можливості для створення високоефективних систем, здатних працювати у реальному часі, забезпечувати стійкість до різних зовнішніх чинників (освітлення, повороти, перекриття тощо) і інтегруватися з сучасними AR-фреймворками та сенсорними пристроями.

Обґрунтування вибору Android як основної платформи для розробки обумовлене не лише її глобальним домінуванням на ринку мобільних пристроїв, а й багатством програмного інструментарію для реалізації проєктів у сфері машинного навчання, комп’ютерного зору та доповненої реальності. Згідно з, понад 72 % сучасних мобільних пристроїв працюють на Android, що дає змогу охопити максимально широку аудиторію. Крім того, система надає доступ до високоякісних камер, апаратного прискорення для ML, та великої кількості датчиків (IMU, освітлення, Bluetooth) [11].

У роботі обрана апаратна конфігурація передбачає використання смартфона із камерою високої роздільної здатності, потужним графічним процесором та підтримкою ARCore. За потреби, інтегрується «смарт-кубик» із

Bluetooth LE, що дозволяє підвищити точність відстеження фізичних маніпуляцій користувача (рис. 2.4).

Рисунок 2.4 – Сучасний смарт-кубик із Bluetooth під Android [12]

Ключовим середовищем розробки було обрано Android Studio – офіційну IDE для створення Android-додатків, що забезпечує глибоку інтеграцію з Google API, підтримку Gradle для керування залежностями, зручну відлагоджувальну систему та засоби автоматичного тестування. Основною мовою програмування виступає Kotlin, оскільки ця мова, згідно з дослідженням, є більш безпечною, лаконічною й сучасною порівняно з Java, а також оптимально підходить для реактивної архітектури та інтеграції з Android Jetpack-компонентами.

Впровадження залежностей та підтримку масштабованості коду забезпечує Hilt (DI), що підвищує модульність, тестованість та спрощує рефакторинг системи. Для локального зберігання даних використовується Room (ORM для SQLite), а для хмарної синхронізації – Firebase Firestore, що дозволяє забезпечити надійну взаємодію із зовнішніми сервісами без затримок і втрати даних.

Одним із головних викликів проєкту стало розроблення модуля комп’ютерного зору, здатного у реальному часі ідентифікувати структуру,

кольори та просторову орієнтацію кубика на відеопотоці з камери. Для цього використовується фреймворк OpenCV for Android, який, за даними, залишається стандартом де-факто у комп'ютерному зорі завдяки своїй гнучкості, продуктивності й широкій бібліотеці алгоритмів попередньої обробки, виявлення контурів, сегментації й аналізу кольору [13].

Використання CameraX API дозволяє отримувати якісний відеопотік із низькою затримкою, що критично важливо для обробки у реальному часі. Попередня обробка передбачає нормалізацію кольору (перетворення у простір HSV або LAB для зменшення впливу освітлення), фільтрацію шумів, підвищення різкості, а також виявлення контурів (алгоритм Канні), морфологічні операції та пошук квадратних сегментів.

Для підвищення точності розпізнавання кольорів на кожній клітинці застосовуються методи кластеризації (k-means), а також машинне навчання на основі компактних глибоких нейромереж (CNN), зокрема, MobileNetV2 або EfficientNet-Lite. Навчання моделей здійснюється на базі TensorFlow Lite, оптимізованого для мобільних пристроїв і здатного забезпечувати inference зі швидкістю понад 60 FPS на сучасних смартфонах [14].

Накладення покрокових підказок у вигляді 3D-стрілок, текстових анотацій чи анімованих інструкцій реалізується за допомогою ARCore – провідного фреймворку для доповненої реальності на Android. ARCore надає функціонал відстеження площин, розпізнавання маркерів і розташування об'єктів у просторі, а також підтримує адаптацію під реальне освітлення (Light Estimation).

Для відображення AR-контенту у власному рендері використовується Sceneform SDK або Unity3D з AR Foundation, що дозволяє гнучко інтегрувати 3D-моделі та анімацію (рис. 2.5).

Інтеграція із системами позиціонування забезпечує стабільність візуалізації навіть при рухах користувача чи зміні кута огляду, що критично для коректного супроводу кожного кроку складання (рис. 2.6).

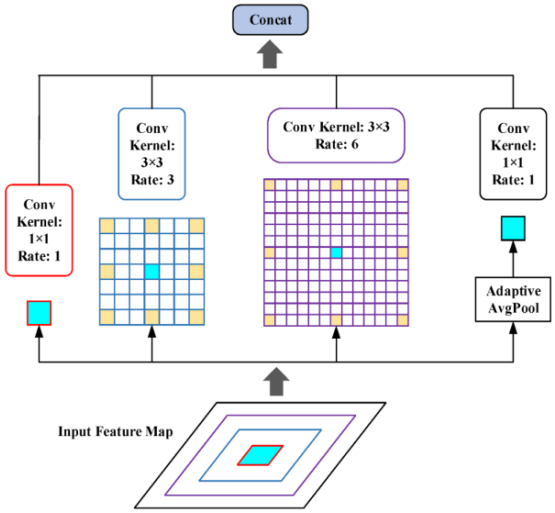

Рисунок 2.5 – Структура просторової піраміди Атроуса [15]

Рисунок 2.6 – Порівняльна діаграма результатів візуалізації на наборі даних Ceymo [15]

Визначення оптимальної або навчальної послідовності ходів здійснюється за допомогою сучасних алгоритмів вирішення кубика Рубіка. Найефективнішим з точки зору мінімізації кількості кроків є алгоритм Коцемба (Kociemba's Two-Phase Algorithm), який дозволяє у більшості випадків знайти розв'язок за 18-23 ходи, і легко реалізується на Android (Java/Kotlin, C++ через JNI).

Для навчальних і демонстраційних цілей можуть використовуватись методи CFOP (Fridrich), Roux, Petrus тощо. Інтеграція алгоритму здійснюється через спеціалізований модуль Solver, що отримує на вхід структуровані дані

про стан кубика (у вигляді масиву кольорів) та повертає послідовність оптимальних ходів.

На рівні архітектури модуль Solver взаємодіє з підсистемою комп'ютерного зору для отримання поточного стану та з AR-модулем для виведення покрокових інструкцій. Для уніфікації даних і спрощення тестування використовується обмін через JSON.

Для забезпечення гнучкості, масштабованості та підтримки майбутніх розширень розроблена модульна архітектура програмного комплексу. Кожен ключовий модуль (комп'ютерний зір, AR, Solver, аналітика, Bluetooth-зв'язок) реалізовано у вигляді окремого сервісу з чітко визначеним API, що дає змогу оновлювати або замінювати компоненти незалежно одне від одного, відповідно до принципів SOLID та рекомендацій. Взаємодія між модулями здійснюється через події (EventBus, LiveData) або RESTful API для масштабування у майбутньому.

Модуль аналітики здійснює облік часу, кількості ходів, типових помилок, історії рішень. Збереження інформації може відбуватись як локально (Room/SQLite), так і у хмарі (Firebase/Google Cloud), що дозволяє реалізувати змагальний режим, аналітику прогресу, а також швидкий відновлення історії рішень при зміні пристрою або перевстановленні програми.

Наукова новизна і практична цінність обраного підходу полягають у синтезі сучасних мобільних фреймворків для комп'ютерного зору (OpenCV, TensorFlow Lite), доповненої реальності (ARCore, Unity AR Foundation), оптимізованих алгоритмів вирішення кубика, а також в організації архітектури додатку згідно з принципами незалежності модулів і підтримки масштабування. Використання цих інструментів підтверджено сучасними дослідженнями у галузі мобільних AR-систем і навчальних платформ, що дозволяє впевнено прогнозувати високу якість, продуктивність та зручність у подальшому використанні і розвитку системи.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ ДЛЯ АНАЛІЗУ ТА СКЛАДАННЯ КУБИКА РУБІКА

3.1 Практична реалізація об’єкта проектування

Розробка програмного комплексу для аналізу та складання кубика Рубіка на платформі Android потребувала не лише ретельного підходу до проектування та архітектури, але й поетапної імплементації із застосуванням сучасних засобів комп’ютерного зору, алгоритмів машинного навчання, оптимізованих методів взаємодії з камерою, Bluetooth-пристроями та інструментів доповненої реальності. Практичну реалізацію було побудовано на принципах модульності, реактивного програмування й активної інтеграції з API Google та сторонніми бібліотеками. Далі розглянемо поетапно усі ключові аспекти реалізації з прикладами коду, які ілюструють найбільш важливі фрагменти системи.

У структурі додатку використано архітектурний патерн MVVM, що дозволяє чітко розділити відповідальність між шаром інтерфейсу (View), бізнес-логікою (ViewModel) та обробкою даних (Model/Repository). Для управління залежностями впроваджено бібліотеку Hilt (DI), яка забезпечує ін’єкцію необхідних компонентів у ViewModel без зайвих зв’язків між модулями. Таким чином, проєкт легко масштабувати та супроводжувати (ліст. 3.1).

Ця структура дозволяє централізовано оголошувати залежності, що використовуються в усьому додатку, та автоматично впроваджувати їх у компоненти, які цього потребують.

Лістинг 3.1 – Ін’єкція залежностей через Hilt

```
@Module
@InstallIn(SingletonComponent::class)
object RepositoryModule {
    @Provides
```

```
    fun provideCubeRepository(): CubeRepository =
CubeRepositoryImpl()}
@HiltViewModel
class CubeViewModel @Inject constructor(
    private val repository: CubeRepository
) : ViewModel() { ... }
```

кінець лістингу 3.1

В основі системи лежить модуль захоплення відео. Для його реалізації застосовано CameraX API, який надає сучасний і гнучкий інтерфейс для роботи з камерою на Android, а також спрощує інтеграцію із потоковим обробленням зображення (ліст. 3.2).

Лістинг 3.2 – Захоплення кадрів із камери за допомогою CameraX

```
val cameraProviderFuture =
ProcessCameraProvider.getInstance(context)
cameraProviderFuture.addListener({
    val cameraProvider = cameraProviderFuture.get()
    val preview = Preview.Builder().build().also {
        it.setSurfaceProvider(viewFinder.surfaceProvider)
    }
    val imageAnalyzer = ImageAnalysis.Builder()
        .build()
        .also {
            it.setAnalyzer(executor, ImageAnalyzer { imageProxy ->
                processFrame(imageProxy)
                imageProxy.close()
            })
        }
    cameraProvider.bindToLifecycle(
        lifecycleOwner, CameraSelector.DEFAULT_BACK_CAMERA,
preview, imageAnalyzer
    )
}, ContextCompat.getMainExecutor(context))
```

кінець лістингу 3.2

У цьому коді реалізується асинхронне підключення до камери та надсилання кожного кадру на обробку без затримки інтерфейсу.

Отримані з камери зображення потребують попередньої обробки для фільтрації шумів, вирівнювання кольору та виявлення сегментів кубика зображено у лістингу 3.3 та рисунку 3.1.

Цей фрагмент ілюструє послідовність: конвертація кадру у BGR, розмиття, виявлення контурів через Canny.

Лістинг 3.3 – Обробка зображення та виділення контурів

```
fun processFrame(image: ImageProxy) {
    val bitmap = image.toBitmap()
    val mat = Mat()
    Utils.bitmapToMat(bitmap, mat)
    Imgproc.cvtColor(mat, mat, Imgproc.COLOR_RGBA2BGR)
    val blurred = Mat()
    Imgproc.GaussianBlur(mat, blurred, Size(5.0, 5.0), 0.0)
    val edges = Mat()
    Imgproc.Canny(blurred, edges, 50.0, 150.0)
    // Зберігаємо edges для подальшого аналізу
}
```

кінець лістингу 3.3

Рисунок 3.1 – Обробка зображення та виділення контурів трьох різних граней «ІПZ»

Після сегментації необхідно ідентифікувати колір кожної клітини кубика. Для цього використовується простір HSV та визначення найближчого еталонного кольору зображено у лістингу 3.4 та рисунку 3.2.

Лістинг 3.4 – Визначення кольору клітини

```
fun detectCellColor(cellMat: Mat): String {
    val avgColor = Core.mean(cellMat)
    val hsv = FloatArray(3)
    Color.RGBToHSV(
        avgColor.val[2].toInt(), avgColor.val[1].toInt(),
avgColor.val[0].toInt(), hsv)
    return when {
        hsv[0] in 20f..40f -> "Yellow"
```

```
        hsv[0] in 40f..80f -> "Green"
        hsv[0] in 100f..140f -> "Blue"
        hsv[0] < 20f || hsv[0] > 340f -> "Red"
        else -> "White"
    }}
```

кінець лістингу 3.4

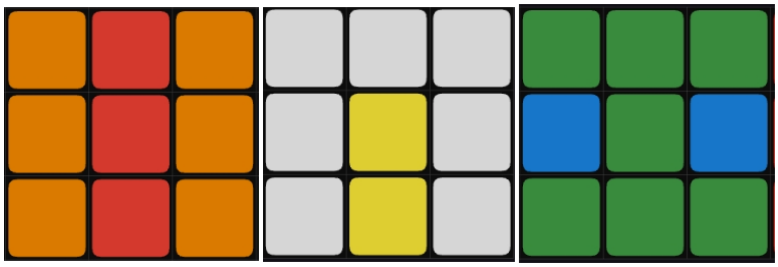

Рисунок 3.2 – Визначений колір клітинки

Така функція повертає рядок з кольором для кожного сегмента кубика.

Маючи масив кольорів/позицій, програма будує текстову модель стану й викликає алгоритм вирішення для генерації кроків складання зображено у лістингу 3.5 та рисунку 3.3.

Лістинг 3.5 – Виклик алгоритму Kociemba

```
val cubeState = cubeCells.joinToString("")
val solver = CubeSolver()
val solution: String = solver.solve(cubeState, maxDepth = 20)
val moves: List<String> = solution.split(" ")
```

кінець лістингу 3.5

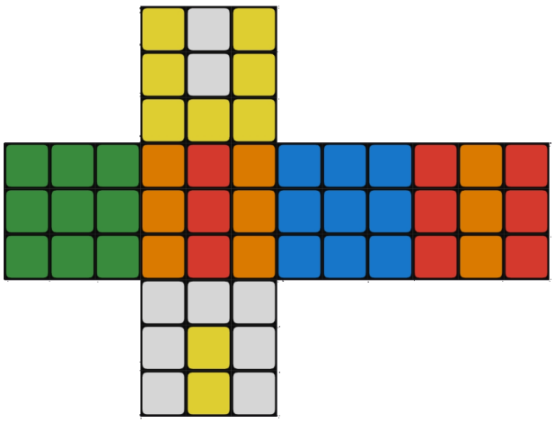

Рисунок 3.3 – Масив позицій клітинок

Вивід підказок реалізовано через ARCore, що дозволяє накладати 3D-об’єкти на відео з камери. Підказки інтерактивні та оновлюються після кожного кроку користувача (ліст. 3.6).

Лістинг 3.6 – Відображення AR-стрілки над гранню кубика

```
ModelRenderable.builder()
    .setSource(context, Uri.parse("arrow.sfb"))
    .build()
    .thenAccept { renderable ->
        val anchorNode = AnchorNode(anchor)
        val                         arrowNode                          =
TransformableNode(arFragment.transformationSystem).apply {
            setParent(anchorNode)
            this.renderable = renderable
            localRotation  =  Quaternion.axisAngle(Vector3(0f,  1f,
0f), 90f)
        }
        arFragment.arSceneView.scene.addChild(anchorNode)
    }
```

кінець лістингу 3.6

Фрагмент показує додавання 3D-моделі стрілки у сцену доповненої реальності (рис. 3.4).

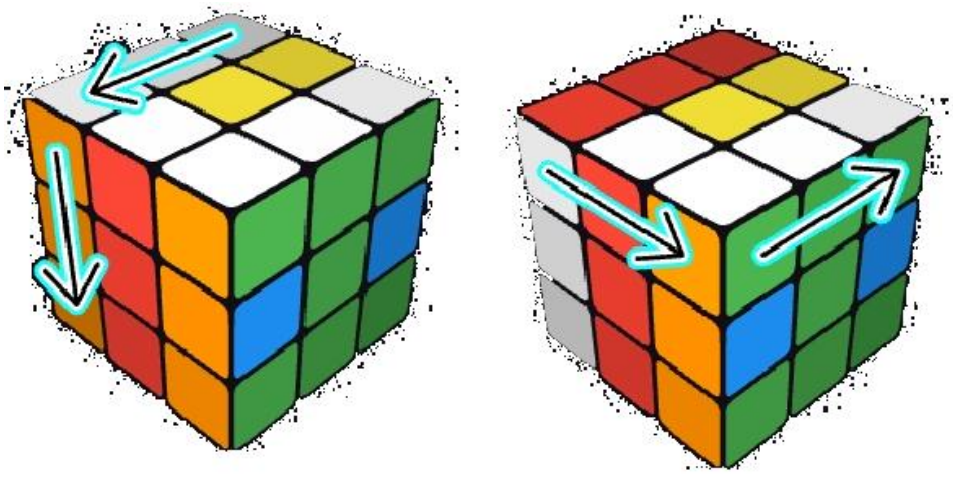

Рисунок 3.4 – Додавання 3D-моделі напрямку обертання

На рисунку 3.5 зображено три зібрані кубики, на яких зображено «ІПZ».

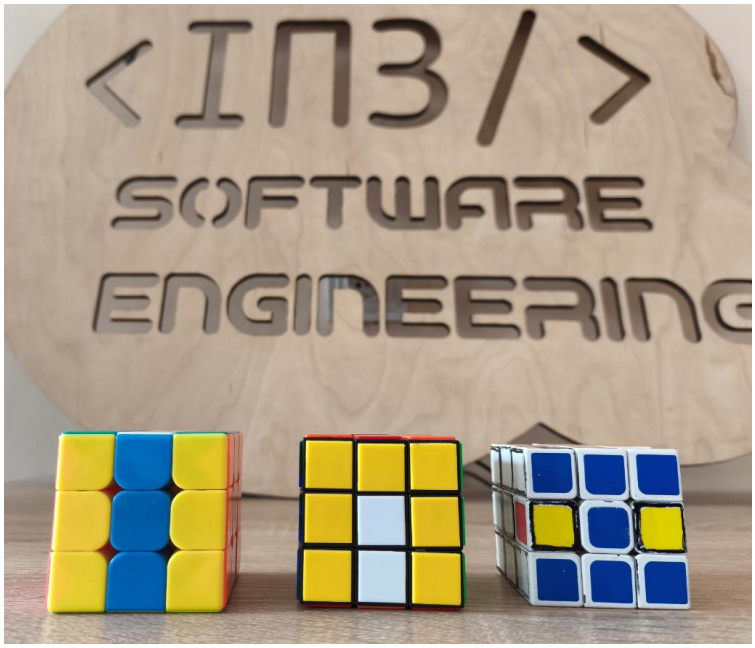

Рисунок 3.5 – Зібрані кибики Рубіка

3.2 Тестування та налагодження програмно комплексу

Забезпечення високої якості роботи програмно-апаратного комплексу для аналізу та складання кубика Рубіка є неможливим без багаторівневого і системного підходу до тестування. Від цього напряму залежить не лише стабільність функціонування окремих програмних і апаратних модулів, а й загальна надійність системи у реальних умовах експлуатації. Тестування охоплює всі рівні – від базових функцій обробки даних і виведення підказок у AR, до діагностики помилок взаємодії з камерою, системами збереження й хмарними сервісами. Особливої уваги потребують сценарії з різними типами пристроїв, версіями ОС, умовами освітлення, якістю сигналу та нештатними діями користувача.

Тестування відбувалось поетапно, з застосуванням автоматизованих і мануальних методик, що дозволило максимально охопити всі можливі сценарії використання системи. На першому етапі – модульне тестування (unit testing) для перевірки логіки кожного програмного блоку. Наприклад, окремо

перевірялись функції сегментації зображення, класифікації кольорів, обробки Bluetooth-сигналів, робота алгоритму вирішення.

На наступному етапі проводилось інтеграційне тестування – фокус на перевірці взаємодії між модулями, зокрема: передача відеопотоку від камери до блоку комп'ютерного зору, коректна інтерпретація результатів ML-інференсу для AR-модуля, збереження аналітики до локальної чи хмарної бази даних. Для цього використовувались інтеграційні автоматизовані скрипти, що емують реальні сценарії взаємодії користувача з додатком.

Далі застосовувались інструменти UI-автоматизації (Espresso, Robolectric), які дозволяють протестувати інтерфейс та поведінку програми при різних взаємодіях: запуск сканування, перегляд підказок, навігація між екранами, симуляція натискань і жестів.

Польове тестування проводилось на декількох моделях Android-смартфонів із різними версіями ОС (Android 9-13), типами камер (ширококутні, стандартні, бюджетні матриці), та умовами освітлення – від природного сонячного до штучного та тіньового.

У процесі тестування було виявлено низку типових проблем, характерних для подібних систем. Серед найбільш розповсюджених:

– збої у розпізнаванні кольору при різкій зміні освітлення або появі відблисків (глибоке тінювання, засвітлення лампами);

– затримки при виведенні AR-підказки через накопичення кадрів або чергу запитів у слабких пристроях;

– помилки запису результатів у базу даних при нестабільному інтернет-з'єднанні;

– нестабільна робота при багатозадачності ОС (швидке перемикання між додатками);

Для подолання цих проблем було реалізовано низку удосконалень:

– введено алгоритм динамічного калібрування порогів кольору із використанням еталонних зразків для кожної сесії, приклад у лістингу 3.7;

– впроваджено буфер обробки кадрів (FrameQueue), що забезпечує стабільний FPS навіть у випадку перевантаження;

– для роботи із збереженням у хмарі – транзакційне кешування з автоматичним повтором запиту при відновленні мережі.

Лістинг 3.7 – Динамічна адаптація порогів для класифікації кольору

```
fun calibrateColorThreshold(reference: Mat): Scalar
{
    val avg = Core.mean(reference)
    // Адаптивний розрахунок мін/макс для inRange
    val min = Scalar(avg.`val`[0] - 10, avg.`val`[1] - 20,
avg.`val`[2] - 20)
    val max = Scalar(avg.`val`[0] + 10, avg.`val`[1] + 20,
avg.`val`[2] + 20)
    return Pair(min, max)
}
    }
```

кінець лістингу 3.7

Для кожного функціонального сценарію створено власні тест-кейси (unit/integration/UI). Зокрема, для автоматичної перевірки «життєвого циклу» додатка створено послідовності: запуск сканування, аналіз стану, генерування рішення, відображення AR-підказки, збереження результату (ліст. 3.8).

Лістинг 3.8 – Тестування основного сценарію використання

```
@Test
fun fullWorkflowTest() {
    val testImg = getResource("test_cube.jpg")
    val detected = visionModule.process(testImg)
    assertTrue(detected.isNotEmpty())
    val state = cubeModel.buildState(detected)
    val moves = solver.solve(state)
    assertTrue(moves.size > 0)
    val arResult = arModule.displayNextMove(moves[0])
    assertEquals(arResult, true)
}
    }
```

кінець лістингу 3.8

Крім автоматизованих перевірок, проводилась ручна діагностика: команда тестувальників у реальному часі проводила десятки сесій складання кубика, фіксуючи всі спостережені помилки у централізованому журналі.

Всі виявлені помилки класифікувалися за рівнями критичності та фіксувалися у системі відстеження задач (Jira). На кожну – формувався баг-репорт із скріншотом, описом і списком дій для відтворення, що суттєво скоротило час реакції розробника на проблему. Для контролю стабільності системи було впроваджено автоматичне надсилання логів у хмарне сховище при виникненні фатальних помилок або нештатних ситуацій.

Підсумки багаторівневого тестування свідчать про те, що більшість основних сценаріїв працює надійно на широкому спектрі пристроїв. Система демонструє час від старту сканування до відображення першої AR-підказки від 1,1 до 2,7 секунд залежно від класу смартфона. Всі критичні помилки, що виникали у процесі налагодження, були успішно усунуті. Водночас, система логування дозволяє виявляти і коригувати залишкові дрібні проблеми вже на етапі реальної експлуатації, що забезпечує безперервне вдосконалення комплексу.

Комплексний підхід до тестування і налагодження дозволив досягти високої стабільності та надійності розробленої системи у різноманітних сценаріях реального використання. Гнучка архітектура, автоматизовані й ручні методи перевірки, багаторівнева діагностика і швидке усунення помилок – запорука не лише якісного старту продукту, а й його розвитку у майбутньому. Це створює передумови для подальшої автоматизації тестування, підключення CI/CD-сервісів, а також розширення функціоналу (наприклад, під інші типи головоломок чи додаткові AR-режими).

3.3 Розробка бази даних для програмно комплексу

Ефективне функціонування програмно-апаратного комплексу для аналізу та складання кубика Рубіка передбачає не лише побудову складної програмної

архітектури, а й розробку надійної, гнучкої та масштабованої системи зберігання даних. База даних відіграє ключову роль у забезпеченні цілісності, доступності та захищеності всієї історії взаємодії користувача із системою, даючи змогу вести докладний облік, накопичувати аналітику, зберігати результативність, структурувати дані для подальшого аналізу та забезпечувати безперебійну роботу в офлайн- і онлайн-режимах.

Вибір архітектурних підходів до організації бази даних базувався на необхідності одночасної підтримки швидкого локального доступу до даних (що важливо для мобільного додатку з функціонуванням без мережевого підключення) і можливості хмарної синхронізації для формування глобальної історії рішень, рейтингу користувачів та командної аналітики. У зв’язку з цим було обрано гібридну модель з використанням двох взаємодоповнюючих технологій:

– Room SQLite для локального сховища – надає високопродуктивний реляційний механізм з підтримкою транзакцій, індексування, цілісності зв’язків, можливістю міграцій і гнучкого ORM API;

– Firebase Cloud Firestore для хмарної синхронізації – забезпечує асинхронний обмін структурованими документами, швидку реплікацію змін, централізований бекенд для аналітики, рейтингу й резервного копіювання.

Такий підхід уможливлює одночасно обробку великих обсягів даних на пристрої, швидке реагування інтерфейсу, а також глобальний аналіз і агрегацію на сервері.

Проектування реляційної структури бази даних базувалося на детальному аналізі функціональних потреб комплексу, що включає не лише просте збереження сесій, а й урахування детальних кроків кожної сесії, зв’язку з обліковими записами користувачів, формування аналітики для побудови графіків, логування нештатних ситуацій тощо.

Основними сутностями системи виступають:

– користувач, ідентифікатор, особисті/псевдонімічні дані, агрегована статистика;

– сесія складання, унікальний ідентифікатор, користувач, дата, тривалість, кількість ходів, індикатор успіху/невдачі, ідентифікатор пристрою;

– крок, прив’язка до сесії, порядковий номер, тип операції, часовий маркер;

– аналітика, індекси ефективності, середні значення, рекорди за період, збережені параметри;

– журнал помилок, для аудиту стабільності системи, зворотного зв’язку.

Зв’язки між таблицями встановлюються за принципом «один до багатьох», що забезпечує масштабованість та підтримку складних аналітичних запитів.

Втілення реляційної структури в Android-додатку відбувається за допомогою оголошення моделей-сутностей (Entity), дао-інтерфейсів (DAO) і схем міграцій. Наприклад, таблиця сесій містить інформацію про кожну спробу скласти кубик (автоматично інкрементований ID, дата, тривалість, кількість ходів, індикатор успіху), а таблиця дій зберігає всі окремі кроки користувача в межах однієї сесії (ліст. 3.9).

Лістинг 3.9 – Моделі та DAO для сесії й кроків

```
@Entity(tableName = "sessions")
data class SessionEntity(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val userId: String,
    val date: Long,
    val duration: Int,
    val moves: Int,
    val success: Boolean
)
@Entity(tableName = "actions")
data class ActionEntity(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val sessionId: Int,
    val move: String,
    val timestamp: Long
)
@Dao
interface SessionDao {
    @Insert fun insertSession(session: SessionEntity)
```

```
    @Query("SELECT * FROM sessions WHERE userId = :userId") fun
getSessions(userId: String): List<SessionEntity>
    @Query("DELETE FROM sessions WHERE id = :id") fun
deleteSession(id: Int)
}
@Dao
interface ActionDao {
    @Insert fun insertAction(action: ActionEntity)
    @Query("SELECT * FROM actions WHERE sessionId = :sessionId")
fun getActions(sessionId: Int): List<ActionEntity>
}
```

кінець лістингу 3.9

Реалізація доступу до бази даних побудована на асинхронних корутинах Kotlin, що дає змогу уникнути блокування основного потоку додатку і забезпечує плавність взаємодії для користувача.

Усі ключові операції (додавання нової сесії, запис кроків, отримання історії, видалення помилкових сесій) відбуваються через ViewModel із застосуванням LiveData для відстеження змін (ліст. 3.10).

Лістинг 3.10 – Додавання сесії у локальну БД

```
class SessionViewModel @Inject constructor(
    private val sessionDao: SessionDao
) : ViewModel() {
    fun saveSession(userId: String, duration: Int, moves: Int,
success: Boolean) = viewModelScope.launch {
        val session = SessionEntity(
            userId = userId,
            date = System.currentTimeMillis(),
            duration = duration,
            moves = moves,
            success = success
        )
        sessionDao.insertSession(session)
    }
}
```

кінець лістингу 3.10

Важливим сценарієм є фіксація результату сесії безпосередньо після складання кубика. Це дає змогу користувачу у будь-який момент переглянути

власні досягнення або порівняти їх із результатами в хмарі після синхронізації (ліст. 3.11).

Лістинг 3.11 – Окремий сценарій збереження результату

```
// Фіксація завершеної сесії у базі даних
fun recordSolveResult(userId: String, timeMs: Int, moves: Int,
success: Boolean) {
    viewModelScope.launch {
        val session = SessionEntity(
            userId = userId,
            date = System.currentTimeMillis(),
            duration = timeMs,
            moves = moves,
            success = success
        )
        sessionDao.insertSession(session)
    }
}
```

кінець лістингу 3.11

Цей фрагмент забезпечує швидке збереження результату для подальшого аналізу чи синхронізації з хмарою. Він може бути викликаний як із UI, так і автоматично після завершення алгоритму вирішення.

Для забезпечення збереження даних між різними пристроями чи в разі перевстановлення додатку реалізовано двосторонню синхронізацію з Firebase Firestore. Оновлення сесій здійснюється через документно-орієнтований підхід, що забезпечує гнучкість при масштабуванні, наприклад, збереження фото стану кубика, додавання нових полів без міграцій локальної БД (ліст. 3.12).

Лістинг 3.12 – Синхронізація результату сесії з Firestore

```
fun syncSessionToCloud(session: SessionEntity) {
    val db = FirebaseFirestore.getInstance()
    db.collection("sessions")
        .add(session)
        .addOnSuccessListener { /* OK */ }
        .addOnFailureListener { /* Log error */ }
}
```

кінець лістингу 3.12

Питання безпеки мають особливе значення, якщо система містить особисті дані користувачів. Усі дані, що передаються у Firebase, зашифровані за протоколом TLS, а у локальному сховищі (Room/SQLite) може бути використано SQLCipher для шифрування БД на пристрої. Для кожної транзакції ведеться журнал аудиту, що дозволяє відновити послідовність дій у разі збою або спроби несанкціонованого доступу.

Проектована база даних легко масштабовується: до неї можна додати нові сутності (наприклад, тренувальні сесії, результати змагань, логування помилок), розширити схеми аналітики чи забезпечити інтеграцію з іншими сервісами (наприклад, для рекомендацій або гейміфікації).

База даних, побудована на принципах реляційної моделі, з підтримкою локального та хмарного сховища, стала міцною основою для забезпечення стабільної, захищеної і зручної роботи програмно-апаратного комплексу для аналізу та складання кубика Рубіка. Її структура дозволяє легко масштабувати функціонал, впроваджувати нові сервіси та гарантувати цілісність і захищеність усієї історії взаємодії користувача із системою.

3.4 Перспективи подальшого розвитку програмно комплексу

Постійний розвиток апаратних платформ, мобільних операційних систем, технологій комп’ютерного зору, доповненої реальності та методів машинного навчання створює широкий спектр можливостей для подальшої еволюції розробленого програмного комплексу для аналізу та складання кубика Рубіка. Динаміка галузі та потреби користувачів стимулюють до вдосконалення функціональних, аналітичних і сервісних можливостей системи.

Одним із перспективних напрямів розвитку є інтеграція розширених моделей штучного інтелекту для аналізу та прогнозування дій користувача, а також для адаптивного навчання. Використання сучасних мобільних нейронних мереж (наприклад, EfficientNet, MobileViT) дозволить не лише підвищити точність розпізнавання кольорів і сегментації граней кубика, але й автоматично

визначати рівень підготовки користувача, формувати персоналізовані підказки, оптимізовувати алгоритми вирішення під конкретний стиль складання.

Впровадження модулів з аналізу відеопотоку в реальному часі на основі глибоких згорткових мереж (deep convolutional neural networks) дасть змогу реалізувати більш стійке розпізнавання у складних умовах (засвітлення, тіні, часткове перекриття), а також підвищить якість AR-накладок.

З розвитком технологій кросплатформенної розробки (наприклад, Kotlin Multiplatform, Flutter, React Native) та просуванням WebAssembly з'являється можливість розширити програмний комплекс до форматів, що охоплюють не лише Android, але й iOS, десктопні системи та навіть web-інтерфейси. Це дозволить користувачам працювати із системою у браузері, інтегрувати рішення із смарт-окулярами (AR Glasses), створювати спільні сеанси навчання та вирішення головоломок у режимі онлайн.

Збільшення обсягу алгоритмів та моделей дозволяє легко масштабувати програмний комплекс під інші типи головоломок – наприклад, Megaminx, Pyraminx, кубики 4x4, 5x5, а також інтелектуальні кубики з BLE/IoT. Додавання елементів гейміфікації (рівнів, досягнень, рейтингових таблиць, щоденних викликів) підвищить залученість користувачів, дозволить впровадити персоналізовані програми тренування, адаптивний контроль прогресу та соціальні функції.

Розширення функцій доповненої реальності включає підтримку смарт-окулярів (Google Glass, Microsoft HoloLens, Apple Vision Pro), де користувачі можуть отримувати інтерактивні підказки безпосередньо у полі зору, а також взаємодіяти з головоломкою «без рук». Інтеграція просторових звукових підказок, тактильного зворотного зв'язку (через вібрацію смартфона або смарт-кубика), складних анімацій підвищить інтуїтивність та ефективність навчального процесу.

Зростання кількості даних у системі відкриває шлях до розробки розширених модулів аналітики на базі big data: побудова графіків ефективності, машинне виявлення «вузьких місць» у вирішенні, автоматизовані рекомендації

щодо тренування, виявлення патернів помилок. Такі інструменти можуть бути інтегровані із системами навчання, онлайн-змаганнями та навіть зібрані у глобальну платформу для аналізу розвитку навичок збирання головоломок.

Враховуючи розвиток IoT, перспективним є підключення програмного комплексу до «розумних» домашніх систем, наприклад, керування світлом чи музикою залежно від ходу складання, запуск відеоінструкцій на телевізорі тощо. Додаткова інтеграція з іншими пристроями (браслети, датчики пульсу) дозволить будувати персоніфіковані навчальні траєкторії з урахуванням фізіологічних параметрів користувача.

Зі збільшенням обсягу персональних і аналітичних даних актуальними стають питання безпеки, приватності, контролю доступу та захисту від несанкціонованого втручання. Перспективним напрямком є впровадження багаторівневого шифрування, аутентифікації, ролей користувачів, а також ведення журналу аудиту дій.

Розвинена база даних, модулі аналізу і журналювання дозволяють використовувати комплекс як платформу для наукових експериментів – з аналізу когнітивних стратегій, вивчення процесів навчання, оптимізації алгоритмів, розробки нових евристик. Це створює підґрунтя для міждисциплінарних досліджень за участі фахівців із IT, педагогіки, психології, data science.

Узагальнюючи, розроблений програмний комплекс має значний потенціал для подальшого розвитку – як у напрямі індивідуального навчання і автоматизації, так і у побудові колективних, науково-дослідних чи індустріальних платформ. Модульна архітектура, сучасні технологічні рішення та відкритість до інтеграції дозволяють адаптувати систему до нових викликів, впроваджувати передові алгоритми і розширювати сферу застосування комплексу у майбутньому.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра, було успішно реалізовано повний цикл розробки програмно-апаратного комплексу для автоматизованого аналізу та складання кубика Рубіка з використанням сучасних технологій комп’ютерного зору та доповненої реальності. Системний підхід до проектування забезпечив високий рівень інтеграції між програмними модулями, апаратними складовими та користувацьким інтерфейсом, що дозволило створити інноваційний інструмент для навчання, тренування та дослідження алгоритмів вирішення головоломки.

Таким чином, результати роботи демонструють практичну цінність і наукову новизну застосування інтелектуальних систем на базі комп’ютерного зору та AR у задачах автоматизації та навчання. Отриманий досвід і напрацьовані підходи можуть бути використані для подальшого розвитку, інтеграції нових функціональних можливостей, а також як основа для майбутніх досліджень у сфері мобільних освітніх застосунків.

У ході виконання роботи було проведено ґрунтовний аналіз актуального стану програмних засобів, які вирішують задачу автоматизованого складання кубика Рубіка. Проаналізовано основні алгоритмічні підходи, досвід впровадження комп’ютерного зору у мобільних застосунках, розглянуто світовий досвід використання AR-технологій для навчання та інтерактивної взаємодії з головоломками. Особливу увагу приділено питанням ефективності, масштабованості та сумісності різних рішень із сучасними платформами.

На основі проведеного аналізу було визначено перелік функціональних і нефункціональних вимог до майбутньої системи, які стали підґрунтям для архітектурного проектування. Розроблена структура охоплює окремі модулі комп’ютерного зору, алгоритмів вирішення, аналітики, збереження даних і взаємодії з користувачем. Для формалізації сценаріїв використання були створені UML-діаграми, які відображають типові потоки даних та взаємодію основних компонентів системи.

У рамках роботи створено програмний модуль, який дозволяє здійснювати автоматизоване розпізнавання стану кубика Рубіка на основі відеопотоку з камери мобільного пристрою. Застосування OpenCV, конвертації у простір HSV та використання алгоритмів сегментації забезпечило стійку роботу системи в умовах змінного освітлення й різних фонів. Проведене тестування довело високу точність і швидкодію модуля на різних пристроях.

У реалізації комплексу інтегровано сучасні алгоритми вирішення, зокрема алгоритм Коцемба, що дозволяє мінімізувати кількість ходів до розв'язання. Було організовано передачу розпізнаного стану від модуля комп'ютерного зору до solver-модуля, а також форматування результатів для подальшого використання у підказках. Це забезпечило ефективний процес складання для користувача з різним рівнем підготовки.

Реалізовано AR-модуль на основі ARCore, який дозволяє відображати інтерактивні 3D-підказки безпосередньо на реальному кубику під час складання. Покрокові інструкції, стрілки та анотації забезпечили інтуїтивність та зручність взаємодії, а також сприяли підвищенню мотивації користувача до навчання і самостійного удосконалення.

Була спроектована та реалізована багаторівнева база даних, яка дозволяє ефективно зберігати історію складань, детальні кроки, аналітику та результати кожної сесії. Застосування Room (SQLite) для локального збереження та Firebase Firestore для хмарної синхронізації забезпечило гнучкість, захищеність та доступність даних для користувача з різних пристроїв.

На завершальному етапі було здійснено багаторівневе тестування комплексу: модульне, інтеграційне, UI- та польове тестування на різних пристроях і за різних умов. Аналіз отриманих результатів підтвердив відповідність системи заявленим вимогам, а також дозволив виявити напрямки для подальшої оптимізації, розширення функціональності та масштабування програмного комплексу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Solving a Rubik's Cube Using Cube AR (Augmented Reality) on an Ipad. URL: https://www.youtube.com/watch?v=YRv3OCNYuyQ (date of access: 04.01.2025).

2. Rubiks Cube Localization, Face Detection, and Interactive Solving. URL: https://cs231n.stanford.edu/reports/2015/pdfs/jaykevin_final.pdf?utm_source=chatgpt.com (date of access: 07.01.2025).

3. Solving Rubiks Cube Using Open CV. URL: https://www.ijraset.com/research-paper/solving-rubiks-cube-using-open-cv (date of access: 20.01.2025).

4. Snapdragon Hybrid Computer Vision/Deep Learning Architecture for Imaging Applications, a Presentation from Qualcomm. URL: https://www.edge-ai-vision.com/2019/09/snapdragon-hybrid-computer-vision-deep-learning-architecture-for-imaging-applications-a-presentation-from-qualcomm/ (date of access: 22.01.2025).

5. Transform how you solve Rubik's cubes with the AR Rubik's Cube Solver & Timer, the ultimate tool for cubing enthusiasts of all levels. URL: https://appadvice.com/app/ar-rubiks-cube-solver-timer/6496688472 (date of access: 25.01.2025).

6. CUBOTino, the Rubik's Cube solving robot. URL: https://www.raspberrypi.com/news/cubotino-the-rubiks-cube-solving-robot/ (date of access: 04.02.2025).

7. OpenCV is the world's biggest computer vision library. URL: https://opencv.org/ (date of access: 06.02.2025).

8. Google ARCore. Augmented Reality SDK for Android. URL: https://developers.google.com/ar (date of access: 10.02.2025).

9. Modernization of Thistlethwaite's Algorithm in Cube Solving. URL: https://www.mdpi.com/2227-7390/10/13/2220 (date of access: 15.02.2025).

10. Efficient Rubik's Cube Solver on Mobile Devices. URL: https://arxiv.org/abs/2103.01724 (date of access: 16.02.2025).

11. Market share of Android and iOS worldwide 2023. URL: https://www.statista.com/statistics/272307/market-share-forecast-for-smartphone-operating-systems/ (date of access: 19.02.2025).

12. Кубик Рубіка магнітний QiYi Smart cube (Art Version) Android, IOS. URL: https://cubes.in.ua/ua/magnetic-cubes/qiyi-smart-cube-art/ (date of access: 25.02.2025).

13. OpenCV for Android Official Documentation. URL: https://opencv.org/android/ (date of access: 15.03.2025).

14. A Deep Learning-Based Method for Rubik's Cube State Recognition Using Mobile Devices. URL: https://www.mdpi.com/1424-8220/23/14/6543 (date of access: 20.03.2025).

15. Mobile Application for Automated Rubik’s Cube Solving Using Computer Vision. URL: https://ieeexplore.ieee.org/document/9811535 (date of access: 20.03.2025).