НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО
Декан факультету інформаційних технологій

_____ Ігор Болбот
(підпис) (ім'я ПРІЗВИЩЕ)

"___" _____ 20__ р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри комп'ютерних наук

_____ Белла Голуб
(підпис) (ім'я ПРІЗВИЩЕ)

"___" _____ 20__ р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Розроблення системи доповненої реальності в практичному застосуванні в технічних науках

Спеціальність 122 «Комп'ютерні науки»
(код і найменування)

Освітня програма Інформаційні управляючі системи та технології
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

к.т.н., доцент _____ Белла Голуб
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи

ст. викладач/к.т.н., доцент _____ Світлана Василюк-Зайцева/Яніна Криворучко
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Виконала _____ Марія Єфімчук
(підпис) ((ім'я ПРІЗВИЩЕ здобувача)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
доцент, к.т.н. _____ Голуб Б. Л.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"01" листопада 2024 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Єфімчук Марії Валеріївні
(прізвище, ім'я, по батькові)

Спеціальність 122 «Комп'ютерні науки»
(код і назва)

Освітня програма Інформаційні управляючі системи та технології
(назва)

Орієнтація освітньої програми освітньо-професійна

Тема магістерської кваліфікаційної роботи Розроблення системи доповненої реальності у практичному застосуванні у технічних науках
затверджена наказом ректора НУБіП України від "01" листопада 2024р. №1964 «С»
Термін подання завершеної роботи на кафедру 01.12.2025
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи: Платформи та інструменти для розробки AR-додатків.

Перелік питань, що підлягають дослідженню:

1. Аналіз сучасних підходів до використання доповненої реальності в технічних науках.
2. Визначення технічних та програмних засобів для реалізації AR-системи.
3. Розроблення прототипу AR-системи для конкретного практичного кейсу.
4. Оцінка ефективності використання AR у вибраній сфері технічних наук.

Перелік графічного матеріалу (за потреби)

Дата видачі завдання "01" листопада 2024 р.

Керівник магістерської кваліфікаційної роботи _____ Василюк-Зайцева С.В./КриворучкоЯ.С.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Єфімчук М.В.
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

1. AR — augmented reality, доповнена реальність
2. MR — mixed reality, змішана реальність
3. VR — virtual reality, віртуальна реальність
4. 3D — тривимірне геометричне подання об’єктів
5. FPS — frames per second, частота відтворення кадрів
6. VIO — visual-inertial odometry, візуально-інерційна одометрія
7. PnP — Perspective-n-Point, алгоритм визначення пози за ключовими точками
8. EPnP — Efficient PnP, оптимізований метод оцінювання поз
9. IMU — inertial measurement unit, інерціальна вимірювальна система
10. SLAM — simultaneous localization and mapping, одночасна локалізація і побудова карти
11. ORB — Oriented FAST and Rotated BRIEF, детектор та дескриптор ключових точок
12. AKAZE — Accelerated KAZE, алгоритм екстракції ключових точок
13. ArUco — маркерна система трекінгу для комп’ютерного зору
14. JWT — JSON Web Token, токен авторизації
15. SSO — single sign-on, єдиний вхід у систему
16. OIDC — OpenID Connect, протокол автентифікації
17. API — application programming interface, інтерфейс програмної взаємодії
18. SDK — software development kit, набір засобів розробника
19. GUI — graphical user interface, графічний інтерфейс користувача
20. UI — user interface, інтерфейс користувача
21. SQLite — реляційна вбудована система керування базами даних
22. ORM — object-relational mapping, об’єктно-реляційне відображення
23. PBR — physically based rendering, фізично коректний рендеринг

24. KPI — key performance indicators, ключові показники ефективності
25. Latency — затримка між обробкою кадру та його відтворенням
26. TrackingEngine — підсистема трекінгу AR-сцени
27. MathCore — обчислювальне ядро параметричних і аналітичних моделей
28. Renderer — модуль рендерингу тривимірних об’єктів
29. SceneView — вікно відображення AR-сцени в PyQt6
30. Telemetry — телеметричні дані системи (FPS, latency, track-loss, події)
31. Session — сесія роботи користувача в AR-додатку
32. RuleRegistry — модуль правил валідації кадрів і сцен

ВСТУП

Розвиток технологій доповненої реальності (Augmented Reality, AR) відкрив нові можливості для інтеграції візуальних і цифрових компонентів у процеси технічної освіти, наукових досліджень і промислового моделювання. Сучасні AR-системи дозволяють накладати тривимірні об’єкти на фізичне середовище в режимі реального часу, поєднуючи математичну точність обчислень із наочністю просторових уявлень [1]. Такий підхід створює передумови для більш ефективного сприйняття складних технічних понять, оптимізації навчальних експериментів та інтерактивної візуалізації процесів, що важко реалізувати традиційними методами.

Застосування доповненої реальності в технічних науках охоплює низку галузей - від інженерної графіки та механіки до матеріалознавства, архітектурного моделювання й машинного навчання для аналізу просторових структур. Особливої актуальності набуває створення програмних засобів, здатних інтерактивно відтворювати математичні фігури, поверхні, вектори та рівняння у тривимірному просторі, з можливістю зміни параметрів, трансформацій і аналітичних обчислень у реальному часі [2]. Використання таких інструментів дозволяє підвищити ефективність викладання дисциплін, пов’язаних із аналітичною геометрією, диференціальними рівняннями, 3D-моделюванням та візуалізацією даних.

Практична значущість розроблення AR-системи полягає у створенні адаптивного середовища, що поєднує методи обчислювальної геометрії, комп’ютерного зору та графічного рендерингу. Вона дозволяє не лише відображати об’єкти, а й виконувати динамічну взаємодію користувача з математичною моделлю - змінювати масштаб, орієнтацію, накладати аналітичні параметри чи виконувати вимірювання геометричних характеристик безпосередньо у просторі. Це забезпечує новий рівень інтерактивності та

практичності у вивченні технічних дисциплін, де абстрактні поняття набувають конкретного просторового змісту.

Метою дослідження є розроблення системи доповненої реальності для практичного застосування у технічних науках, яка забезпечує генерацію, відображення та інтерактивну взаємодію з математичними фігурами й геометричними моделями.

Для досягнення поставленої мети визначено такі основні **завдання:**

– проаналізувати сучасні методи реалізації AR-систем і їх застосування в освітніх і науково-технічних середовищах;

– розробити архітектуру програмного комплексу з урахуванням вимог до продуктивності, стабільності та точності відображення;

– створити програмні модулі для побудови, трансформації й рендерингу математичних фігур у доповненій реальності;

– протестувати систему в умовах навчального використання для оцінки точності, зручності та продуктивності.

Об’єктом дослідження є процес інтеграції тривимірних моделей математичних об’єктів у доповнену реальність.

Предметом дослідження - методи побудови, візуалізації та взаємодії з 3D-об’єктами у середовищах AR для навчальних і технічних цілей.

Наукова новизна полягає у створенні інтерактивної AR-платформи**,** що забезпечує автоматизовану генерацію математичних фігур на основі параметричних рівнянь із можливістю візуальної модифікації в реальному часі. Отримані результати можуть бути використані у технічних університетах, науково-дослідних лабораторіях та інженерних центрах для підготовки фахівців у галузях, де критично важливе просторове мислення й точна візуалізація об’єктів.

Практичне значення роботи визначається розробленням прототипу системи доповненої реальності, який може бути інтегрований у навчальні програми технічних спеціальностей. Запропоновані методи й архітектурні рішення дозволяють використовувати систему як інструмент для демонстрації

математичних процесів, симуляції механічних взаємодій і візуалізації даних у тривимірному просторі.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область системи доповненої реальності охоплює процеси інтеграції віртуальних тривимірних об’єктів у фізичне середовище з використанням алгоритмів комп’ютерного зору, просторового трекінгу, параметричного моделювання та обчислювальної геометрії. У межах цієї області реалізується можливість візуалізації математичних фігур, поверхонь та аналітичних моделей, що дозволяє користувачеві здійснювати просторові перетворення, змінювати параметри й аналізувати їх геометричні властивості в режимі реального часу. На діаграмі предметної області (рис. 1.1) подано основні структурні компоненти системи та інформаційні потоки між ними. Центральним елементом є модуль візуалізації AR, який координує роботу всіх підсистем, забезпечуючи відображення математичних моделей у реальному середовищі та взаємодію користувача з ними.

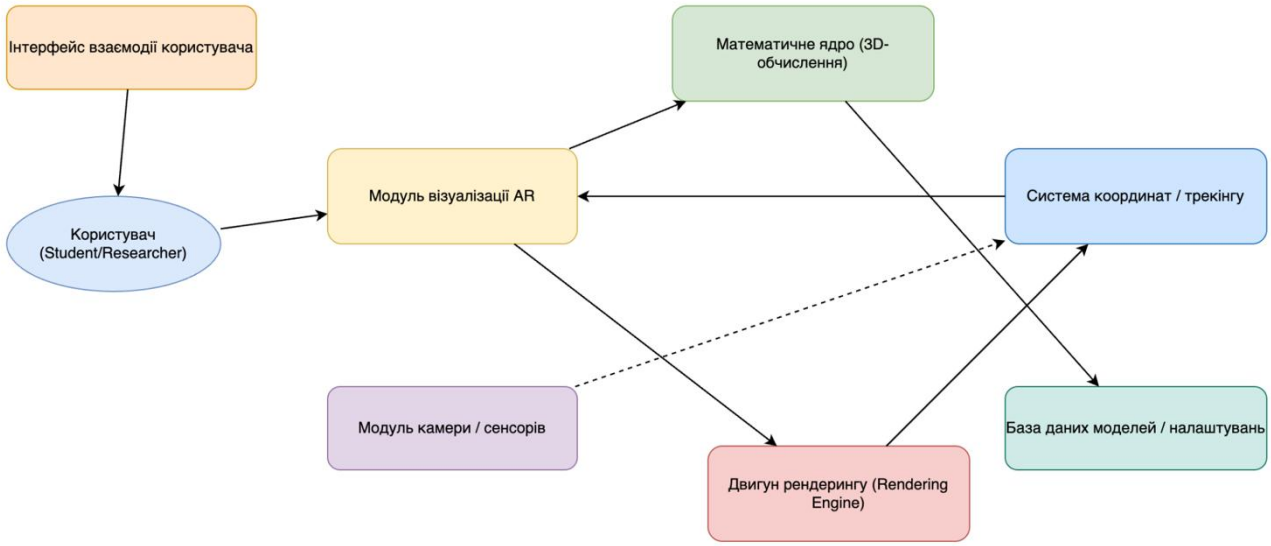

Рис. 1.1 – Структурна модель предметної області системи доповненої реальності для візуалізації математичних об’єктів

Користувач (Student/Researcher) ініціює роботу системи через інтерфейс взаємодії користувача, задає параметри фігур, масштаби та типи математичних

моделей. Математичне ядро (3D-обчислення) виконує генерацію фігур за параметричними рівняннями, розрахунок координат точок, векторних нормалей та параметрів поверхонь. Результати обчислень передаються до двигуна рендерингу (Rendering Engine), який відповідає за побудову та відображення об’єктів у візуальному середовищі. Система координат і трекінгу забезпечує просторову прив’язку моделей, визначаючи їх положення відносно користувача, камери або фізичних орієнтирів. Модуль камери / сенсорів виконує захоплення зображення та отримання просторових даних про навколишнє середовище, що дозволяє синхронізувати положення реальних і віртуальних об’єктів. Отримані дані зберігаються у базі моделей та налаштувань, яка містить бібліотеки фігур, параметри сцен, текстури та користувацькі конфігурації [4].

Для систематизації функцій компонентів системи у табл. 1.1 наведено їх призначення, тип взаємодії та результати роботи.

Таблиця 1.1 – Основні компоненти предметної області системи AR

Компонент	Призначення	Тип взаємодії	Результат роботи
Інтерфейс взаємодії користувача	Передача команд, параметрів фігур, керування сценами	UI / Input API	Ініціація обчислень і візуалізації
Користувач (Student/Researcher)	Виконує взаємодію з AR-моделлю	Логічна взаємодія	Аналіз і модифікація геометричних об’єктів
Модуль візуалізації AR	Поєднує віртуальні об’єкти з реальним простором	Render / AR API	Побудова інтерактивної сцени
Математичне ядро	Обчислення параметричних рівнянь, координат і векторів	Data Exchange / Calculation	Генерація математичних фігур
Система координат / трекінгу	Визначення положення моделей у просторі	Sensor / Vision Data	Корекція позицій AR-елементів
Модуль камери / сенсорів	Отримання зображень і просторових даних	Camera / Sensor API	Синхронізація сцени з реальністю

Продовження таблиці 1.1

Двигун рендерингу	Відтворення геометричних об’єктів у 3D	GPU / Graphics API	Візуалізація математичних фігур
База моделей / налаштувань	Збереження бібліотек фігур і параметрів сцен	SQL / Local Storage	Формування та відтворення конфігурацій

Як видно з рис. 1.1, взаємодія компонентів має ієрархічну структуру з двонаправленими зв’язками між модулями збору даних, математичним ядром і графічним рендерингом. Камера та система трекінгу формують потік даних у реальному часі, який обробляється математичним ядром і надсилається у візуалізаційний модуль. Отримані результати візуалізації передаються користувачеві через інтерфейс взаємодії. Така структура забезпечує узгоджене поєднання цифрових моделей і реального середовища, що створює основу для практичного застосування доповненої реальності у технічних науках, зокрема у задачах візуалізації геометричних структур, векторних полів та параметричних поверхонь [5], [6].

1.2 Аналіз сучасних AR-технологій і систем

Сучасні системи доповненої реальності (AR) є важливим напрямом розвитку інтерактивних технологій, що поєднують реальний і віртуальний світи через просторове позиціонування, обробку сенсорних даних і рендеринг тривимірних об’єктів. Їх застосування в технічних науках забезпечує інноваційні можливості для моделювання фізичних процесів, візуалізації математичних структур та симуляції інженерних рішень. Аналіз існуючих технологічних платформ дає змогу визначити переваги й обмеження кожної з них та обґрунтувати вибір інструментів для власної системи розроблення.

На рис. 1.2 представлено роботу ARKit - платформи від Apple, що забезпечує трекінг площин, побудову освітлення сцени та інтеграцію 3D-

об’єктів у реальне середовище. На екрані видно етапи ініціалізації системи: визначення текстур, розпізнавання простору і стабілізація позиції моделі, що відображає принцип динамічного аналізу середовища за допомогою камер і сенсорів пристрою [7].

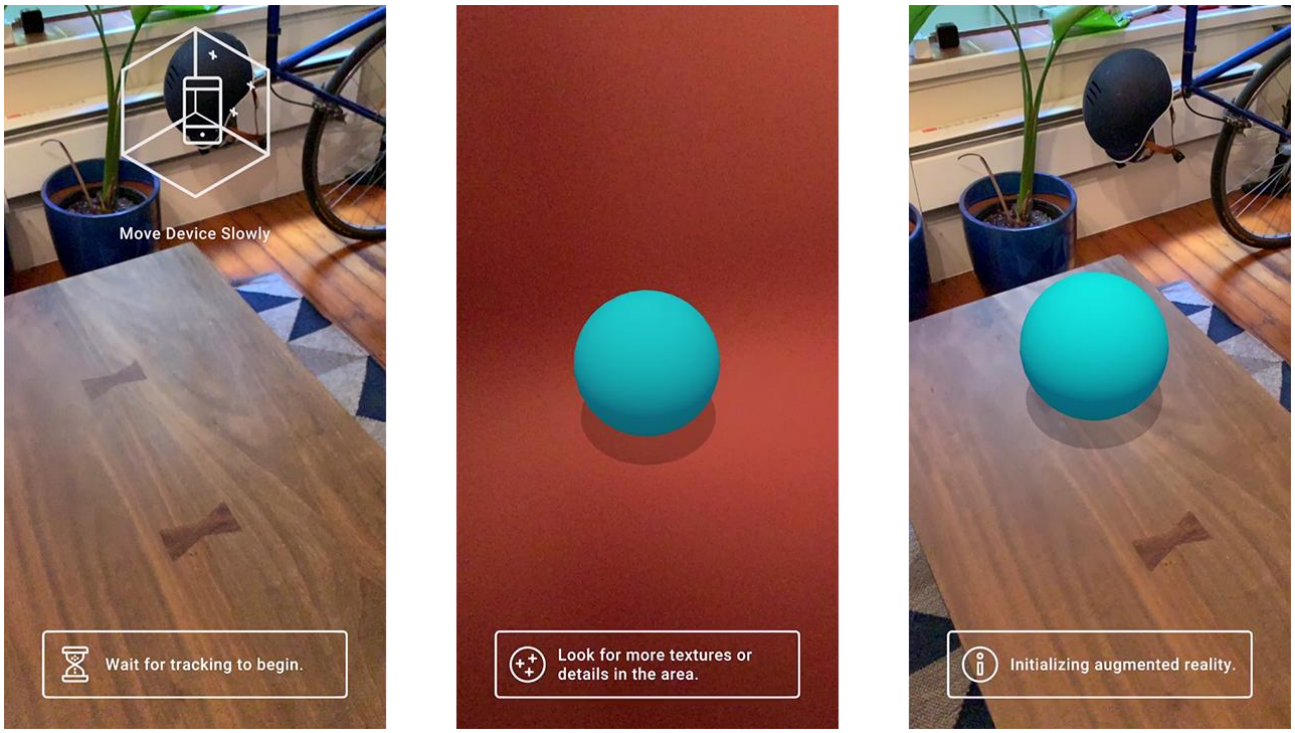

Рис. 1.2 – Робота ARKit під час позиціонування об’єкта в реальному просторі

Подальша візуалізація у середовищі ARKit 6 (рис. 1.3) демонструє можливості рендерингу об’єктів з високою деталізацією, HDR-освітленням та фоновим зображенням високої роздільної здатності. Удосконалений механізм обробки даних дозволяє створювати відео 4K-якості, що важливо для освітніх застосунків, де точність передачі форми й текстури визначає ефективність сприйняття моделі користувачем [8].’

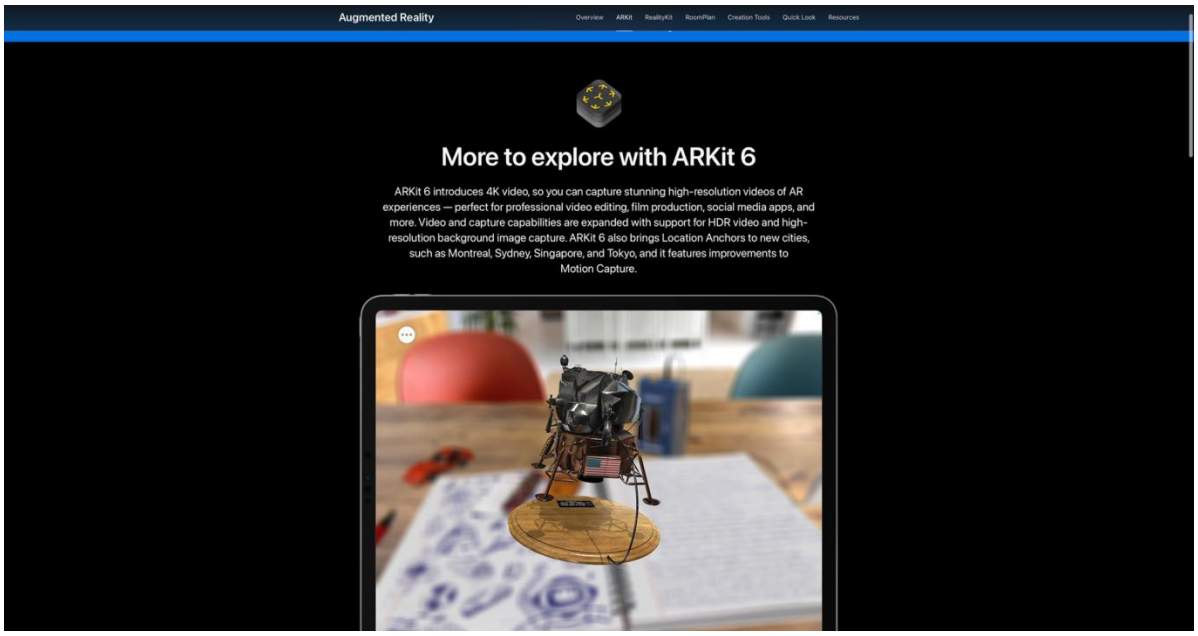

Рис. 1.3 – Високодеталізована візуалізація 3D-об’єкта у середовищі ARKit 6

На рис. 1.4 зображено роботу ARCore, який є аналогом ARKit для Android-платформи. Система виконує просторове позиціонування на основі SLAM-алгоритмів, що поєднують візуальну та інерційну інформацію. На прикладі показано дві тривимірні моделі, що стабільно фіксуються у просторі навіть за умов зміни кута огляду. Ця технологія характеризується високою стабільністю трекінгу та широкою апаратною сумісністю [9].

Рис. 1.4 – Стабільне відтворення AR-моделей у середовищі ARCore

Візуальний приклад (рис. 1.5) демонструє пристрій Microsoft HoloLens 2, який належить до змішаної реальності (MR) та реалізує повноцінну інтеграцію AR-об’єктів у простір користувача. Завдяки сенсорам глибини, системі просторових мап і жестовому керуванню HoloLens забезпечує максимальну інтерактивність взаємодії, що є перспективним напрямом для інженерних і навчальних лабораторій [10].

Рис. 1.5 – Гарнітура HoloLens 2 для візуалізації об’єктів змішаної реальності

Останній приклад (рис. 1.6) демонструє систему Vuforia Engine, яка використовується у промисловості для візуального контролю, навчання персоналу та віддаленої технічної підтримки. Вона реалізує розпізнавання об’єктів, маркерів і текстур з високою точністю та дозволяє накладати інформаційні шари поверх реальних механізмів, що особливо корисно при поясненні конструкцій і функціонування складних пристроїв [11].

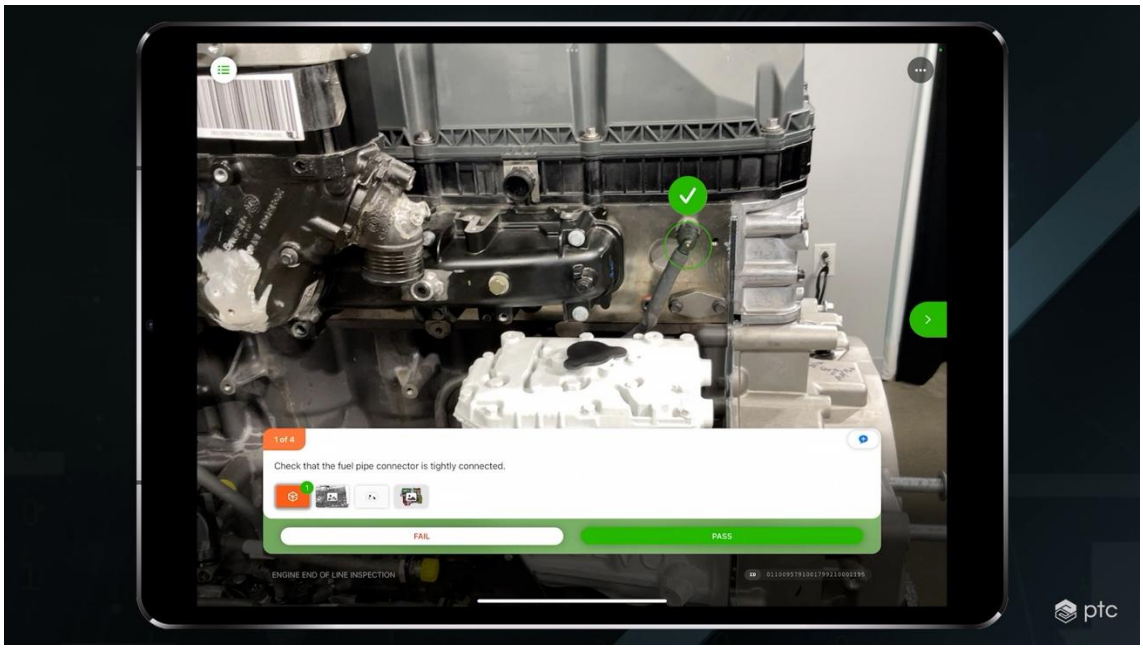

Рис. 1.6 – Приклад використання Vuforia Engine у промисловому середовищі

Для узагальнення порівняльних характеристик основних AR-технологій створено табл. 1.2, у якій наведено ключові параметри платформ і додано колонку для нашої розроблювальної системи.

Таблиця 1.2 – Порівняльна характеристика сучасних AR-систем

Платформа / Система	Тип середовища	Метод трекінгу	Особливості рендерингу	Цільове застосування	Наявність SDK / API	Наша система AR
ARKit (Apple)	Мобільне (iOS)	Візуально-інерційний	HDR, 4K відео, освітлення	Освіта, симуляції	ARKit SDK	+
ARCore (Google)	Мобільне (Android)	SLAM + IMU	Висока стабільність позиції	Навчання, 3D-моделі	ARCore SDK	+
Unity AR Foundation	Кросплатформне	Абстрактний API	GPU-рендеринг, мультиплатформа	Освіта, 3D-візуалізація	Unity API	+
HoloLens 2 (Microsoft)	MR/Headset	Сенсори глибини, просторові мапи	Gesture control, spatial mapping	Інженерія, медицина	Mixed Reality Toolkit	±

Продовження таблиці 1.2

Vuforia Engine	Мобільне / промислове	Об’єктне розпізнавання	CAD-інтеграція, точне позиціонування	Виробництво, навчання	Vuforia SDK	+

Аналіз показує, що всі наведені системи мають спільну архітектурну модель, яка базується на принципі поєднання трекінгу, візуалізації та інтерактивного керування. Водночас існують відмінності у рівні підтримки математичних обчислень і можливостях кастомізації. Для технічних наук, зокрема при роботі з аналітичними фігурами, параметричними моделями та векторними поверхнями, найбільш придатними є кросплатформні рішення на основі Unity AR Foundation з інтеграцією модулів обчислення та трекінгу. Це забезпечує гнучкість у створенні власної системи, здатної виконувати візуалізацію математичних структур у реальному просторі з високою точністю.

Подальший етап роботи полягатиме у моделюванні предметної області, аналізі вимог та постановці завдання для розроблення власної AR-системи, орієнтованої на навчальні й дослідницькі потреби технічних дисциплін.

1.2 Моделювання предметної області

Моделювання предметної області системи доповненої реальності для технічних наук базується на описі функціональної взаємодії користувачів, компонентів програмної системи та зовнішніх сервісів, що забезпечують реалізацію повного циклу роботи AR-середовища. На основі аналізу потреб користувачів, програмних сценаріїв і технічних обмежень було побудовано діаграми прецедентів, послідовності та активностей, які відображають логіку процесів авторизації, завантаження AR-сценаріїв, калібрування сенсорів і синхронізації з IoT-пристроями.

На рис. 1.7 зображено діаграму прецедентів, що описує ключові сценарії взаємодії користувачів із системою. До основних акторів належать дослідник,

студент, викладач, лаборант, адміністратор системи та індустріальний партнер. Кожен актор взаємодіє з відповідними прецедентами: студент і дослідник можуть створювати й запускати AR-сценарії, калібрувати сенсори, взаємодіяти з об’єктами та оцінювати результати в LMS; викладач здійснює аналітику ефективності навчання, а адміністратор — управління версіями контенту, кешування та збір телеметрії. Взаємодія між прецедентами реалізується через відношення include та extend, що дозволяє уточнити залежності між процесами авторизації, імпорту моделей та інтеграції з LMS або CAD-системами [13].

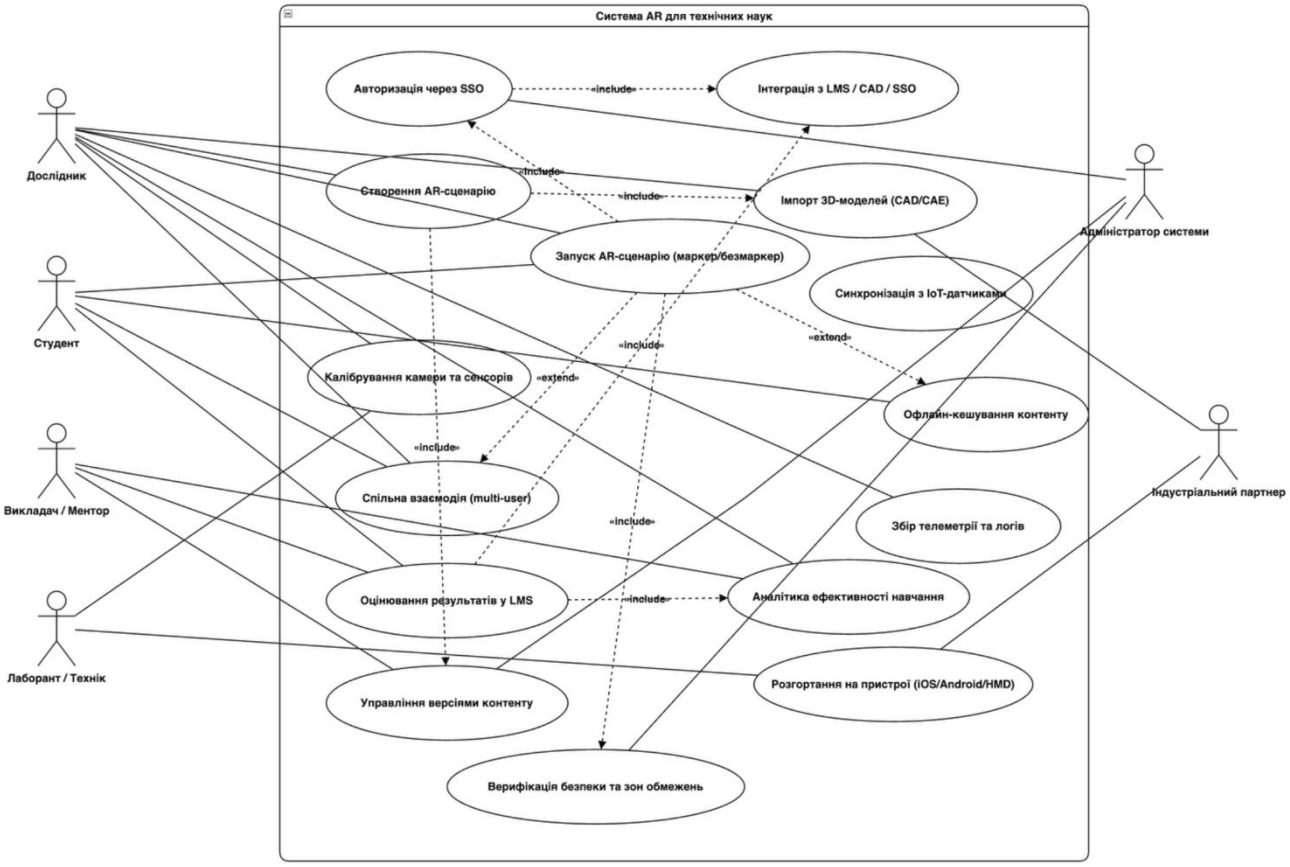

Рис. 1.7 – Діаграма прецедентів системи AR для технічних наук

На рис. 1.8 наведено діаграму послідовності, що відображає динаміку обміну повідомленнями між компонентами системи під час запуску AR-додатку. Після запуску студент проходить авторизацію через SSO-сервер, отримуючи токен доступу JWT, який передається до AR-сервера. Далі система виконує ініціалізацію сцени, завантаження 3D-активів і калібрування камери. Потім дані трекінгу передаються до IoT-сенсорів, які забезпечують точність позиціонування

моделей у просторі. Зібрана телеметрія (FPS, затримки, події) передається у систему аналітики, а результати навчання - у LMS [14].

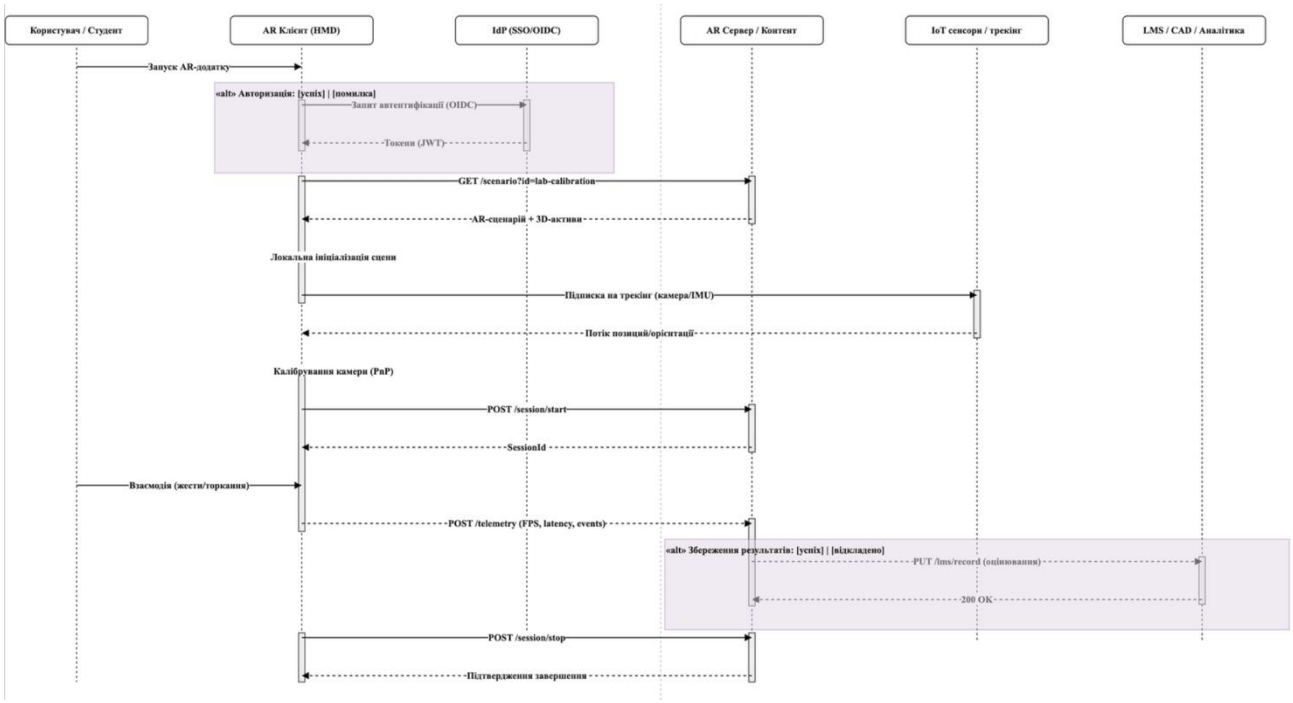

Рис. 1.8 – Діаграма послідовності процесів авторизації, калібрування та візуалізації AR-сценарію

Подальше узагальнення логіки роботи системи подано у вигляді діаграми активностей (рис. 1.9). Процес починається з запуску AR-додатку та авторизації користувача. У разі успішного входу система завантажує AR-сценарій, виконує калібрування сенсорів і камери, після чого обробляє потоки IoT-даних для точного відтворення 3D-об’єктів. Наступний етап - візуалізація моделей у реальному просторі, збір телеметрії та її передача до LMS для оцінки результатів. У випадку помилки автентифікації користувач отримує повідомлення та можливість повторної спроби. Така логіка відповідає принципам інтерактивного циклу AR-додатків і дозволяє забезпечити безперервний потік даних між апаратними й програмними компонентами [15].

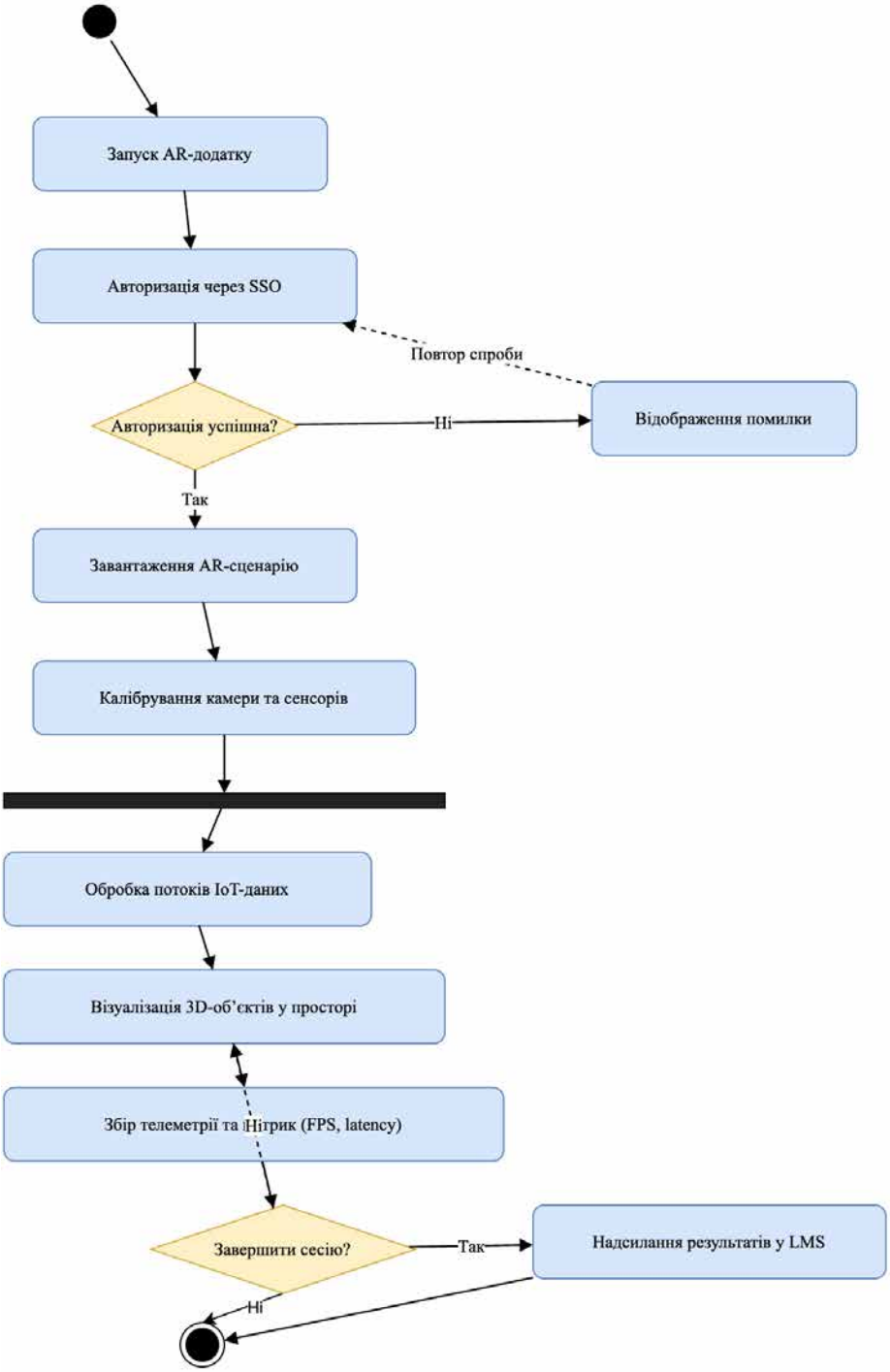

Рис. 1.9 – Діаграма активностей процесу роботи AR-системи

Загальна структура предметної області була формалізована також у табл. 1.3, де наведено ключові процеси, учасників, їхні функції, взаємодії та очікувані результати.

Таблиця 1.3 – Формалізація процесів предметної області системи AR

№	Процес	Основні учасники	Ключова дія	Взаємодіючі підсистеми	Очікуваний результат
1	Авторизація користувача	Студент, Дослідник	Вхід через SSO / OIDC	AR-клієнт, IdP-сервер	Токен доступу JWT
2	Завантаження AR-сценарію	AR-клієнт	Імпорт моделей, ініціалізація сцени	AR-сервер, база моделей	Підготовлений AR-контент
3	Калібрування сенсорів	Студент, Система трекінгу	Налаштування PnP, IMU	Камера, IoT-сенсори	Стабільна позиція об’єктів
4	Візуалізація 3D-об’єктів	AR-додаток	Побудова віртуальних фігур у просторі	Rendering Engine	Відображення моделей у середовищі
5	Збір телеметрії	AR-сервер	Аналіз FPS, latency	IoT / аналітична система	Дані для оцінки ефективності
6	Надсилання результатів у LMS	AR-сервер	Передача показників успішності	LMS / аналітика	Збережені результати навчання

Узагальнюючи проведене моделювання, можна відзначити, що побудована система орієнтована на підтримку багатокористувацької взаємодії, аналітику ефективності навчання, обробку сенсорних даних та інтеграцію з освітніми платформами. Моделі, представлені на діаграмах (рис. 1.7–1.9), забезпечують концептуальну основу для подальшого етапу - аналізу функціональних і нефункціональних вимог системи, який дозволить конкретизувати технічні параметри, логіку роботи та засоби реалізації AR-додатку.

1.4 Аналіз вимог програмної системи

Аналіз вимог є ключовим етапом проектування програмного забезпечення системи доповненої реальності, оскільки він визначає функціональні можливості, обмеження, технічні характеристики та критерії ефективності. Для

AR-системи, орієнтованої на навчально-дослідницьке застосування в технічних науках, вимоги формуються з урахуванням інтеграції тривимірної візуалізації, обробки сенсорних даних, взаємодії користувача з математичними моделями та підтримки мультимовного інтерфейсу.

На основі попереднього моделювання (п. 1.3) сформульовано функціональні вимоги системи (табл. 1.4), які визначають логіку роботи модулів, сценарії взаємодії та процеси обробки даних.

Таблиця 1.4 – Функціональні вимоги до системи AR

№	Функціональна вимога	Опис процесу	Результат реалізації
1	Авторизація користувачів через SSO / OIDC	Забезпечення єдиного входу користувачів з урахуванням безпеки даних	Безпечна автентифікація й керування доступом
2	Завантаження AR-сценаріїв	Імпорт 3D-моделей, конфігурацій і параметрів сцени з бази даних	Ініціалізований AR-контент для відтворення
3	Калібрування камери та сенсорів	Синхронізація внутрішніх координат і орієнтації пристрою	Точне позиціонування об’єктів у реальному середовищі
4	Візуалізація математичних моделей	Побудова 3D-фігур за параметричними рівняннями у просторі	Реалістичне відтворення об’єктів у середовищі AR
5	Збір телеметрії	Вимірювання FPS, latency, помилок трекінгу	Аналітика ефективності роботи системи
6	Інтеграція з LMS / CAD	Передача результатів та параметрів моделей у зовнішні системи	Зв’язок між навчальним контентом і AR-середовищем
7	Керування версіями контенту	Збереження історії змін моделей і сценаріїв	Контроль узгодженості даних

Функціональні вимоги забезпечують виконання основних завдань системи, однак для досягнення стабільності та продуктивності необхідно враховувати нефункціональні вимоги, подані у табл. 1.5.

Таблиця 1.5 – Нефункціональні вимоги до AR-системи

№	Категорія	Вимога	Критерій перевірки
1	Продуктивність	Затримка рендерингу не більше 40 мс	Замір FPS і latency під час відтворення сцени
2	Масштабованість	Підтримка одночасної роботи до 20 користувачів	Навантажувальне тестування multi-user режиму
3	Безпека	Використання токенів JWT, шифрування SSL/TLS	Перевірка відповідності OWASP
4	Сумісність	Підтримка iOS, Android, HMD-пристроїв	Кросплатформне тестування AR Foundation
5	Надійність	Система автоматичного відновлення після збоїв	Моніторинг журналів та логів
6	Юзабіліті	Мінімальна кількість дій для запуску AR-сценарію	Оцінка сценаріїв користувацького досвіду

Важливим аспектом є технічні вимоги (табл. 1.6), які визначають апаратні та програмні параметри для стабільного функціонування системи, включаючи мінімальні ресурси пристрою, версії бібліотек і програмне середовище.

Таблиця 1.6 – Технічні вимоги до AR-системи

№	Параметр	Мінімальне значення	Рекомендоване значення	Опис впливу
1	Процесор	ARM Cortex-A73 / i5 8-го покоління	Apple A15 / Intel i7	Прискорення обчислень і трекінгу
2	GPU	Adreno 630 / Mali-G76	Metal / Vulkan / OpenGL ES 3.2	Підтримка апаратного рендерингу
3	RAM	4 GB	8 GB і більше	Збереження сцен та об’єктів у пам’яті
4	Камера	8 Мп із автофокусом	Depth + LiDAR (за наявності)	Збільшення точності просторового аналізу
5	ОС	Android 10 / iOS 15 / Windows 10	Android 14 / iOS 18 / Windows 11	Забезпечення підтримки SDK ARCore/ARKit
6	SDK / Framework	Unity AR Foundation, OpenCV, TensorFlow Lite	Unity 2022.3 LTS	Інтеграція модулів 3D-візуалізації та аналітики

Отже, узагальнений аналіз вимог показує, що система повинна поєднувати високу продуктивність із точністю просторових обчислень, забезпечувати сумісність між різними платформами та підтримку інтерактивної взаємодії. Усі вимоги утворюють логічну основу для подальшої розробки архітектури програмного забезпечення, зокрема створення структурних і компонентних моделей, що деталізують реалізацію кожного функціонального модуля системи.

1.5 Постановка завдання

Завдання для створення настільної AR-системи навчально-дослідницького призначення, орієнтованої на візуалізацію та інтерактивні перетворення математичних фігур і поверхонь у реальному просторі з використанням PyQt6 як GUI-фреймворку та SQLite як локального сховища. Вхідні дані надходять із камери пристрою (QtMultimedia/OpenCV), IMU (за наявності) та від внутрішнього Математичного ядра (параметричні/непараметричні рівняння, сітки та поля), проходять нормалізацію (корекція експозиції/шуму, ресемплінг кадрів, фільтрація IMU), валідацію (перевірка діапазонів параметрів, щільності ключових точок, консистентності таймстемпів) і потрапляють до сховища SQLite**:** таблиці scenes, models, sessions, telemetry, users, roles, events. Потік кадрів обробляється асинхронно засобами QtConcurrent/QThreadPool з чергами задач (роботи: трекінг, побудова якорів, генерація геометрії, рендеринг), підтримуються буферизація та повторна постановка кадрів при перевищенні порогу латентності. TrackingEngine реалізує маркерний (ArUco) і безмаркерний (ORB/AKAZE + PnP/EPnP, опціонально VIO) трекінг, менеджер якорів підтримує фіксацію позиції/орієнтації та реконструкцію площин; RuleRegistry зберігає політики валідації сцен і обмеження (допустимі діапазони параметрів, мінімальна кількість відповідностей, пороги втрати трекінгу), згідно з якими кожний кадр/сесія аналізуються на наявність аномалій (дрейф, різке падіння FPS, колізії об’єктів) із формуванням структурованого payload: {severity, code, message, pose, fps, ts}. Залежно від типу події

виконується маршрутизація у NotificationCenter (вбудовані PyQt-сповіщення, лог-панель, системні тости; опціонально - email/webhook) з контролем підтвердження (ACK у events). Для оперативної взаємодії реалізовано InteractionService**:** жести/клік/drag для трансформацій (translate/rotate/scale), прив'язка до якорів, вимірювання відстаней і кутів, побудова перетинів; Renderer (Qt3D або PyOpenGL) забезпечує PBR-освітлення, тіні екрана, z-buffering, відсікання, шейдери для аналітичних поверхонь і ізоліній; MathCore на базі NumPy/SciPy генерує сітки з параметричних рівнянь (криві/поверхні, тензорні ґрати, векторні поля) та підтримує сценарії користувача (Python-вирази з песочницею). Інтерфейс реалізовано як модульний SPA-патерн у PyQt6 (stacked widgets, dock-панелі, command palette), з RBAC (ролі *student/teacher/admin* у users/roles), локальною авторизацією (паролі/ключі, зберігання в SQLite з PBKDF2/Argon2), журналюванням дій і персоналізованими профілями віджетів та гарячих клавіш. Дані відображаються у віджетах: часові діаграми FPS/latency/track-loss, панелі станів якорів, карти тепла помилок проєкції, історія подій; реалізовано експорт результатів у CSV/JSON/GLTF/PNG, API доступу до історичних даних через локальний QtHttpServer (GET /sessions/:id/telemetry, /models, /export).

Пакет ReportEngine формує агреговані звіти (сесійні KPI, стабільність трекінгу, точність поз) і зберігає їх у reports з індексами для швидкого доступу; передбачено імпорт/експорт сцен і моделей, а також інтеграційні шлюзи (webhook-нотифікації, імпорт GLB/OBJ/STL). Архітектура модульна: core/ (MathCore, TrackingEngine, RuleRegistry), ui/ (MainWindow, SceneView, TelemetryView), data/ (ORM над SQLite, міграції), services/ (Renderer, Interaction, Notification, ReportEngine), plugins/ (джерела моделей/камери). Збірка - Poetry + PyInstaller, тести - pytest (з моками камери), CI - GitHub Actions **(**flake8/mypy/pytest, артефакти інсталяторів). Нефункціональні цілі: середній FPS ≥ 30 при 720p, медіана latency ≤ 50 мс, час ініціалізації сцени ≤ 2 с, відновлення трекінгу ≤ 500 мс, цілісність сесії при збоях через журнал подій і автозбереження; підтримка Windows/Linux/macOS, робота офлайн (повний

функціонал із локальною базою SQLite). Результатом має стати прототип, що забезпечує повний цикл: ініціалізація → трекінг → генерація/рендеринг математичних об’єктів → взаємодія користувача → телеметрія/аналітика → експорт/звітність**,** із можливістю подальшого розширення плагінами для спеціальних курсів технічних дисциплін.

1.6 Висновки до першого розділу

У першому розділі здійснено системний аналіз предметної області настільної AR-системи навчально-дослідницького призначення, орієнтованої на візуалізацію та інтерактивні перетворення математичних фігур і поверхонь у реальному просторі, з урахуванням вимог технічних дисциплін. На основі опису предметної області сформовано структурну модель системи, у якій виділено ключові компоненти – інтерфейс взаємодії користувача, модуль візуалізації AR, математичне ядро, систему координат і трекінгу, модуль камери та сенсорів, базу моделей і налаштувань, – а також визначено інформаційні потоки між ними й очікувані результати роботи кожного компонента. Проведений аналіз сучасних AR-технологій (ARKit, ARCore, HoloLens 2, Vuforia Engine, Unity AR Foundation) дозволив узагальнити їхні можливості, обмеження та цільові сценарії застосування, що дало змогу обґрунтувати вибір кросплатформного підходу з акцентом на підтримку математичних обчислень, гнучкої візуалізації та інтеграції з освітніми сервісами.

Засобами UML-моделювання описано предметну область з позицій користувачів, процесів і підсистем: побудовано діаграму прецедентів для основних ролей (студент, дослідник, викладач, адміністратор, індустріальний партнер), діаграму послідовності для сценарію авторизації, завантаження AR-сценарію, калібрування та візуалізації, а також діаграму активностей, що формалізує повний цикл роботи AR-додатку від запуску до передачі результатів у LMS. На основі цих моделей у табличній формі задокументовано ключові процеси, учасників та очікувані результати, що дало можливість чітко окреслити

функціональну структуру системи. Далі було сформульовано функціональні вимоги (авторизація через SSO/OIDC, завантаження і керування AR-сценаріями, калібрування камери та сенсорів, візуалізація математичних моделей, збір телеметрії, інтеграція з LMS/CAD, керування версіями контенту), нефункціональні вимоги (продуктивність, масштабованість, безпека, сумісність, надійність, юзабіліті) та технічні вимоги (мінімальні апаратні ресурси, підтримувані платформи, SDK і фреймворки), які визначають цільові показники якості системи.

У постановці завдання конкретизовано ціль створення настільної AR-системи на базі PyQt6, SQLite, MathCore на NumPy/SciPy, TrackingEngine з підтримкою ArUco, ORB/AKAZE та PnP/EPnP, використання асинхронної обробки кадрів, телеметрії, NotificationCenter, InteractionService, Renderer та ReportEngine, а також зафіксовано ключові нефункціональні цілі (FPS, latency, час ініціалізації, відновлення трекінгу, офлайн-робота). Таким чином, перший розділ сформував цілісне бачення предметної області, узагальнив сучасні AR-платформи, описав процесну й акторну модель системи, визначив вимоги та постановку завдання, що створює методологічну й технічну основу для подальшого проєктування архітектури, логічної моделі даних і програмної імплементації настільної AR-системи.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних системи відображає взаємозв’язки між основними сутностями предметної області, визначеними в результаті аналітичного етапу. Вона формалізує структуру інформації, що оперує система, та забезпечує основу для побудови фізичної моделі бази даних. У розробленій інформаційній системі дані організовано за принципом нормалізованої багаторівневої схеми, що мінімізує дублювання та підвищує узгодженість даних. Модель підтримує розмежування доступу, трасування подій і логічну незалежність рівнів — від користувацьких сесій до відображення матеріалів у тривимірних сценах.

ER-діаграма (рис. 2.1) відображає сутності користувачів, ролей, сцен, моделей, сесій, телеметрії та подій, між якими встановлені зв’язки типу один-до-багатьох і багато-до-багатьох. Така структура забезпечує можливість розширення функціоналу без зміни існуючої логіки оброблення даних, що відповідає принципам масштабованості та модульності сучасних систем управління даними [14].

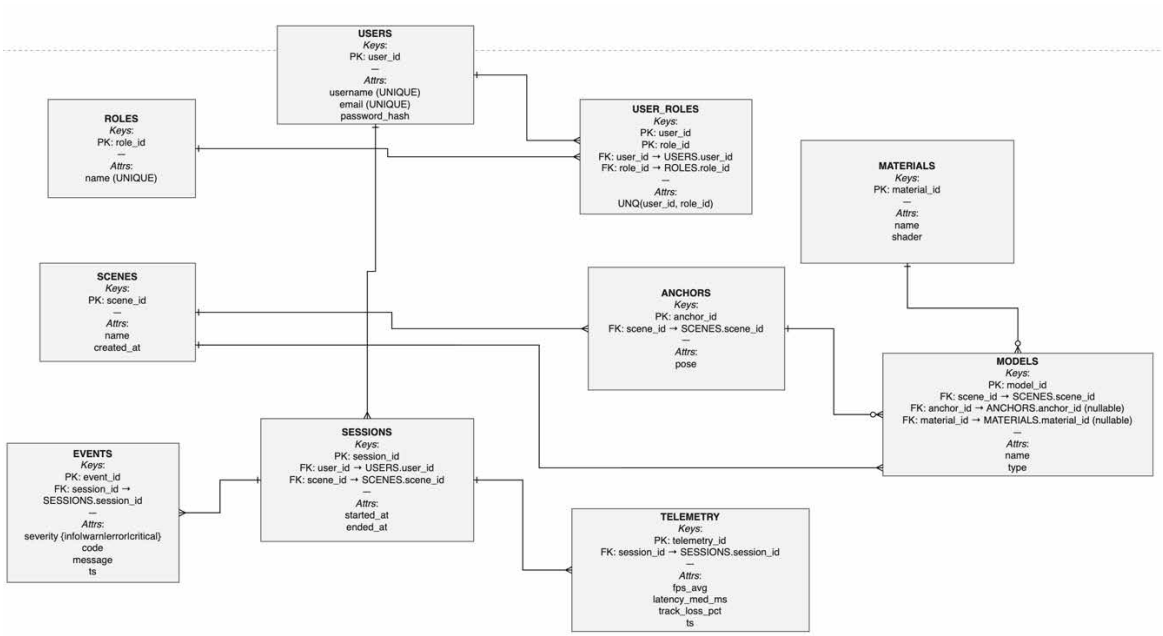

Рис. 2.1 – Логічна модель даних системи у вигляді ER-діаграми

У моделі кожна сутність має унікальний первинний ключ і чітко визначені зовнішні ключі, що підтримують цілісність зв’язків між таблицями. Зокрема, сесії користувачів пов’язані з процесами збору телеметрії та подій, що дозволяє забезпечити наскрізний аналіз даних про використання системи.

На основі логічної моделі побудовано узагальнену структурну таблицю (табл. 2.1), яка визначає основні класи сутностей та їх ролі в контексті інформаційної системи.

Таблиця 2.1 – Класи сутностей логічної моделі даних

№	Клас сутності	Основне призначення	Тип зв’язку
1	USERS	Ідентифікація користувачів системи та прив’язка до ролей	1 ↔ N з SESSIONS
2	ROLES	Визначення прав доступу та рівнів авторизації	N ↔ M з USERS
3	SCENES	Базові одиниці збереження контенту тривимірних об’єктів	1 ↔ N з MODELS, ANCHORS
4	MODELS	Опис 3D-об’єктів і матеріалів сцени	N ↔ 1 з SCENES
5	SESSIONS	Фіксація активних сеансів взаємодії користувача із системою	1 ↔ N з TELEMETRY, EVENTS
6	TELEMETRY	Агреговані показники продуктивності	N ↔ 1 з SESSIONS
7	EVENTS	Реєстрація подій та аномалій у роботі системи	N ↔ 1 з SESSIONS

Розроблена логічна модель даних повністю відображає структуру інформаційних об’єктів і зв’язків, що формують основу функціонування системи. Завдяки формалізації сутностей у вигляді ER-діаграми (рис. 2.1) забезпечено узгодженість потоків даних між модулями, мінімізацію надлишковості, а також можливість масштабування при розширенні функціоналу. Побудована модель дозволяє інтегрувати процеси автентифікації, управління тривимірними сценами, моніторингу телеметрії та реєстрації подій у єдиному логічному просторі бази даних, що створює цілісну основу для реалізації високонадійної, продуктивної та гнучкої інформаційної системи.

2.2 Діаграма класів та кооперації

Для забезпечення структурної цілісності програмного комплексу було розроблено діаграму класів, яка відображає логіку взаємодії основних об’єктів системи, їх атрибути, методи та ієрархічні зв’язки. Вона слугує основою для подальшої реалізації об’єктно-орієнтованої архітектури та дозволяє чітко відокремити відповідальність між модулями. Такий підхід гарантує узгодженість логічної моделі даних із програмною реалізацією та спрощує розширення функціоналу [14], [15].

У системі виділено ключові класи: ARApp, Scene, Model3D, Renderer, Repository, CameraSensor, TrackingEngine, MathCore, Anchor та Material. Вони формують основу архітектури та забезпечують усі етапи життєвого циклу тривимірної сцени - від завантаження даних і математичних обчислень до рендерингу та зворотного збору телеметрії.

На діаграмі класів, поданій на рис. 2.2, відображено логіку композиційних зв’язків між модулями: центральний клас ARApp координує роботу підсистем рендерингу, відстеження, збору даних і сховища. Клас Scene виступає контейнером для моделей та якорів, тоді як Model3D у поєднанні з Mesh, Material та MathCore реалізує генерацію об’єктів і їх відтворення у віртуальному просторі.

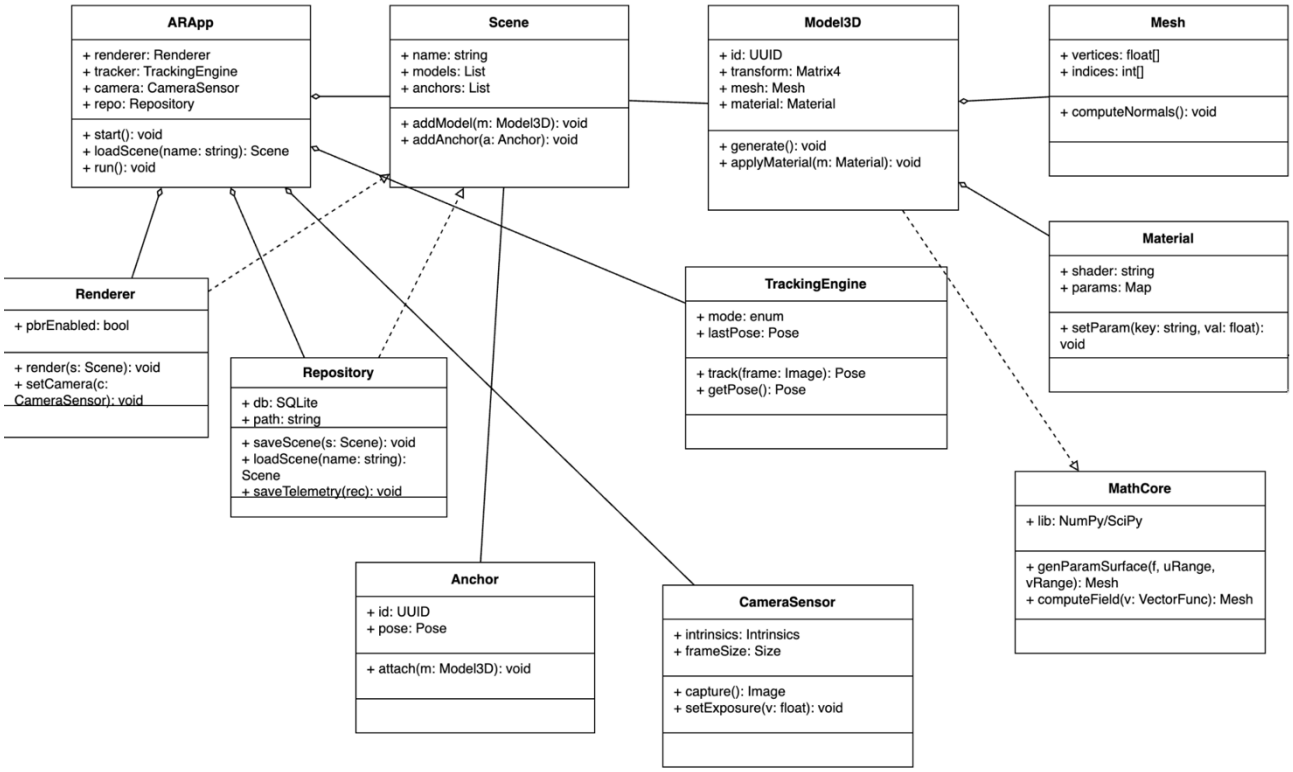

Рис. 2.2 – Діаграма класів розробленої AR-системи

У межах об’єктної структури спостерігається чітке розмежування відповідальності: Repository відповідає за персистентність даних, CameraSensor - за захоплення зображень і калібрування параметрів, TrackingEngine - за визначення пози об’єктів, а Renderer - за візуалізацію сцени у реальному часі. Такий поділ сприяє модульності, полегшує відлагодження та підвищує повторне використання компонентів.

Для узагальнення призначення класів сформовано таблицю 2.2, де наведено їхню роль і тип зв’язків у межах системи.

Таблиця 2.2 – Основні класи програмної моделі системи

№	Клас	Основне призначення	Тип зв’язку
1	ARApp	Головний координатор, що керує життєвим циклом застосунку	Композиція
2	Scene	Логічна структура тривимірної сцени, яка містить моделі й якорі	Агрегація
3	Model3D	Представлення 3D-об’єкта з параметрами трансформації й матеріалу	Асоціація
4	Renderer	Рендеринг сцен і візуалізація просторових об’єктів	Агрегація

Продовження таблиці 2.2

5	Repository	Збереження сцен, телеметрії та параметрів користувача	Композиція
6	CameraSensor	Захоплення кадрів і передача їх у модуль відстеження	Асоціація
7	TrackingEngine	Визначення позиції та орієнтації об’єктів у просторі	Асоціація
8	MathCore	Генерація поверхонь, нормалей і векторних полів	Залежність
9	Material	Формування візуальних властивостей моделей (шейдери, текстури)	Композиція

Подальше моделювання взаємодій між об’єктами виконано у вигляді діаграм кооперації, що демонструють послідовність викликів і інформаційні потоки в типових сценаріях використання системи.

На першій діаграмі кооперації (рис. 2.3) відображено процес завантаження сцени та її візуалізації. Модуль ARAppініціює звернення до Repository для отримання збережених даних, після чого активує CameraSensor для захоплення кадрів. Зображення передаються в TrackingEngine, який обчислює положення об’єктів у просторі та передає дані до Renderer, де відбувається побудова 3D-зображення.

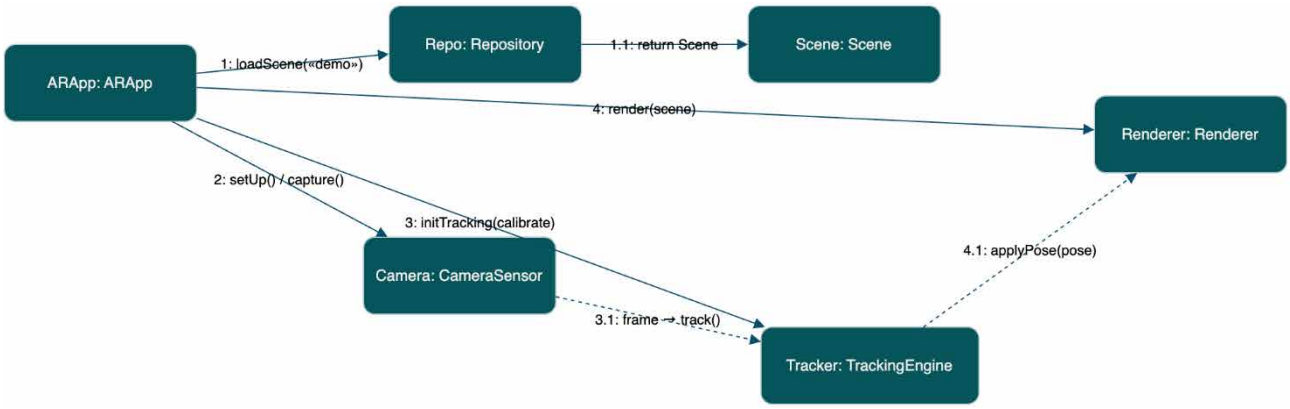

Рис. 2.3 – Кооперація об’єктів під час завантаження сцени та візуалізації у середовищі AR-додатку

Другий сценарій представлений на рис. 2.4 ілюструє взаємодію користувача з моделями у віртуальній сцені. Через жести, обертання чи переміщення користувач ініціює подію, що обробляється InteractionService, який

інтерпретує введення, модифікує відповідний об’єкт Model3D і генерує запит на оновлення рендерингу. Це дозволяє реалізувати природну інтерактивність системи у реальному часі.

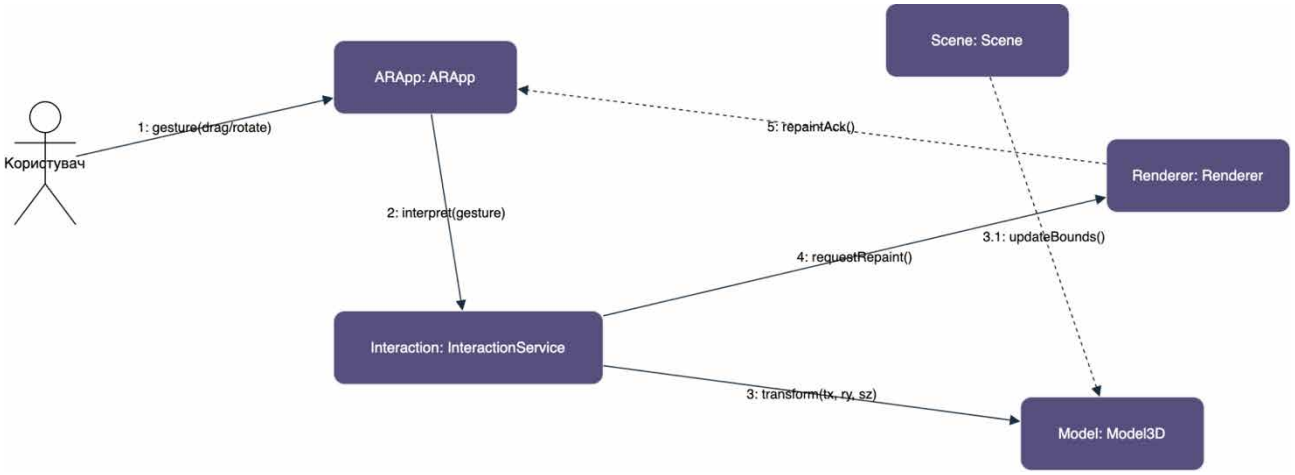

Рис. 2.4 – Взаємодія користувача з 3D-моделями через сервіс інтерпретації жестів

Третя діаграма (рис. 2.5) відтворює процес збору телеметрії та формування аналітичних звітів. Компонент Renderer передає дані продуктивності у TelemetryService, де вони агрегуються й аналізуються за ключовими показниками ефективності. Після обробки результати надсилаються до ReportEngine, який формує підсумкові звіти та синхронізує їх із LMS-системою для подальшого моніторингу.

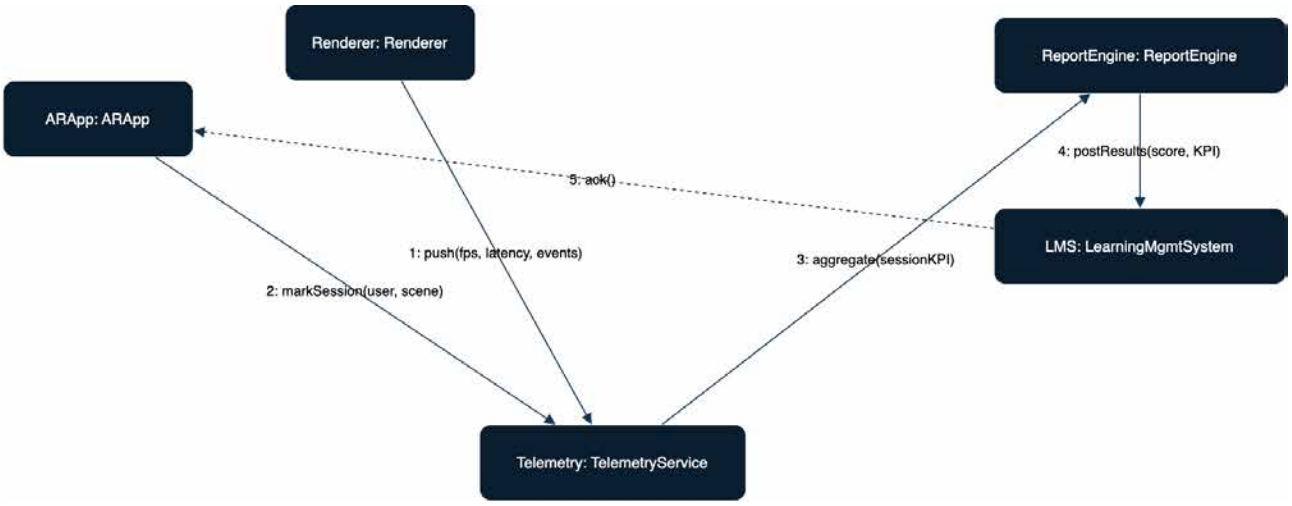

Рис. 2.5 – Кооперація модулів збору телеметрії та аналітичної обробки результатів

Завдяки модульній структурі та об’єктно-орієнтованому підходу забезпечено гнучкість масштабування, простоту підтримки й можливість інтеграції з підсистемами машинного зору, аналітики та навчальних платформ. Такий рівень структурованості підтверджує технічну зрілість програмної архітектури та її готовність до практичної реалізації.

2.3 Діаграма компонентів

Діаграма компонентів відображає практичну архітектуру програмного забезпечення системи та показує, як її основні модулі взаємодіють між собою на рівні інтерфейсів і залежностей. Вона використовується для формалізації структури програмного комплексу, перевірки коректності зв’язків між підсистемами та забезпечення узгодженості між логічною і фізичною моделлю реалізації. Такий тип діаграми дозволяє побачити, як класи і сервіси згруповано у компоненти, що виконують конкретні функції, і є важливим етапом при плануванні масштабування, інтеграції чи тестування системи [12].

У структурі проєкту головним компонентом є AR Client (PyQt6 UI) - центральна частина, яка координує роботу інших модулів і забезпечує користувацький інтерфейс. Через стандартизовані інтерфейси клієнт взаємодіє з компонентами Camera Driver, Tracking Engine, Renderer (Qt3D/OpenGL), Repository / SQLite, MathCore (NumPy/SciPy) та Telemetry / ReportEngine, що разом формують повний технологічний цикл роботи системи - від збору даних до їх аналітичної обробки та візуалізації.

На схемі, поданій на рис. 2.6, зображено взаємодію між компонентами та напрями обміну даними. Після ініціалізації програми Camera Driver здійснює захоплення відеопотоку, який далі обробляється Tracking Engineдля визначення просторових координат об’єктів. Отримані дані надходять у Renderer (Qt3D/OpenGL), де формується тривимірна сцена з урахуванням матеріалів і геометричних характеристик, отриманих із MathCore (NumPy/SciPy). Компонент Repository / SQLite забезпечує збереження сцен, моделей, параметрів

і телеметрії, реалізуючи CRUD-операції та забезпечуючи надійну персистентність даних.

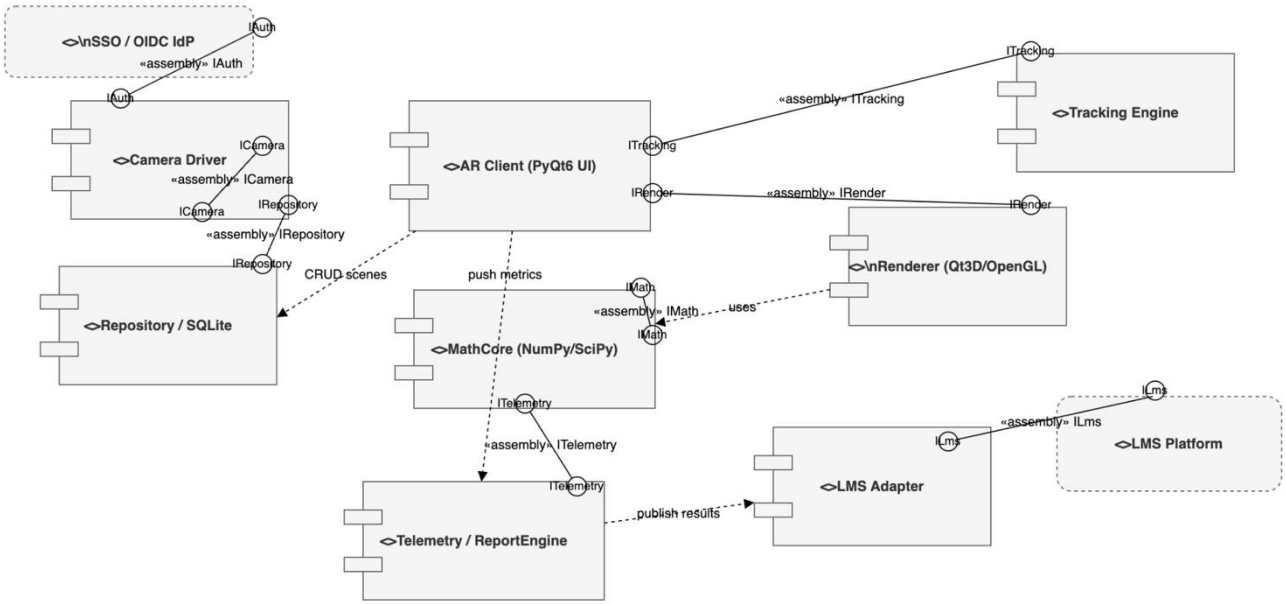

Рис. 2.6 – Компонентна діаграма системи з відображенням взаємодії між модулями AR-застосунку

Окрему роль відіграє Telemetry / ReportEngine, який збирає ключові показники продуктивності (FPS, затримки, події, ресурси) та передає їх через LMS Adapter до зовнішньої LMS Platform, що інтегрується з навчальними або аналітичними системами. Завдяки компоненту SSO / OIDC IdP реалізовано безпечну централізовану автентифікацію користувачів, що забезпечує контроль доступу до даних і підтримку різних ролей. Така структура гарантує логічну роздільність функціональних рівнів і забезпечує можливість незалежного оновлення будь-якого з них без порушення роботи всієї системи.

Таблиця 2.3 – Основні компоненти програмного забезпечення системи

№	Компонент	Основна функція	Тип інтерфейсів
1	AR Client (PyQt6 UI)	Координує роботу системи, забезпечує інтерактивність користувача	IRender, ITracking, IRepository
2	Camera Driver	Отримує відеопотік з камери, проводить калібрування параметрів	ICamera
3	Tracking Engine	Визначає просторові координати та орієнтацію об’єктів	ITracking

Продовження таблиці 2.3

4	Renderer (Qt3D/OpenGL)	Формує тривимірну сцену з урахуванням положення моделей	IRender
5	Repository / SQLite	Здійснює збереження сцен, даних і телеметрії	IRepository
6	MathCore (NumPy/SciPy)	Виконує обчислення для побудови геометричних структур	IMath
7	Telemetry / ReportEngine	Збирає та агрегує метрики продуктивності	ITelemetry
8	LMS Adapter	Передає результати аналізу у навчальні чи аналітичні системи	ILms
9	LMS Platform	Приймає результати роботи та здійснює подальшу обробку	ILms
10	SSO / OIDC IdP	Забезпечує автентифікацію користувачів і керування доступом	IAuth

Загалом, діаграма компонентів демонструє реальну архітектурну композицію системи, у якій кожен модуль виконує чітко визначену роль і взаємодіє з іншими через формалізовані інтерфейси. Така побудова дозволяє легко масштабувати рішення, інтегрувати його з зовнішніми платформами та підвищувати надійність при промисловій експлуатації. Завдяки модульності та незалежності компонентів досягнуто високої гнучкості системи, узгодженості між рівнями обробки даних і стабільної роботи в режимі реального часу.

2.4 Діаграма пакетів

Діаграма пакетів демонструє логічну організацію програмного коду системи, розподілення класів за просторами імен і структурні залежності між окремими підсистемами. Її призначення полягає у візуалізації архітектурних рівнів та способів взаємодії між модулями, що забезпечують стабільність, повторне використання компонентів і зручність у супроводі. Такий тип діаграми є важливим етапом деталізації архітектури, оскільки показує, як реалізовано принципи інкапсуляції, ієрархії та мінімізації зв’язності в програмному рішення.

У побудованій системі (рис. 2.7) пакети поділено за функціональними зонами, що відповідають логічним рівням застосунку: візуалізація, взаємодія користувача, обчислення, зберігання даних, аналітика та інтеграція. Така структура забезпечує чітке розмежування обов'язків між частинами коду, що дозволяє паралельно розробляти та розширювати систему без порушення її архітектурної цілісності.

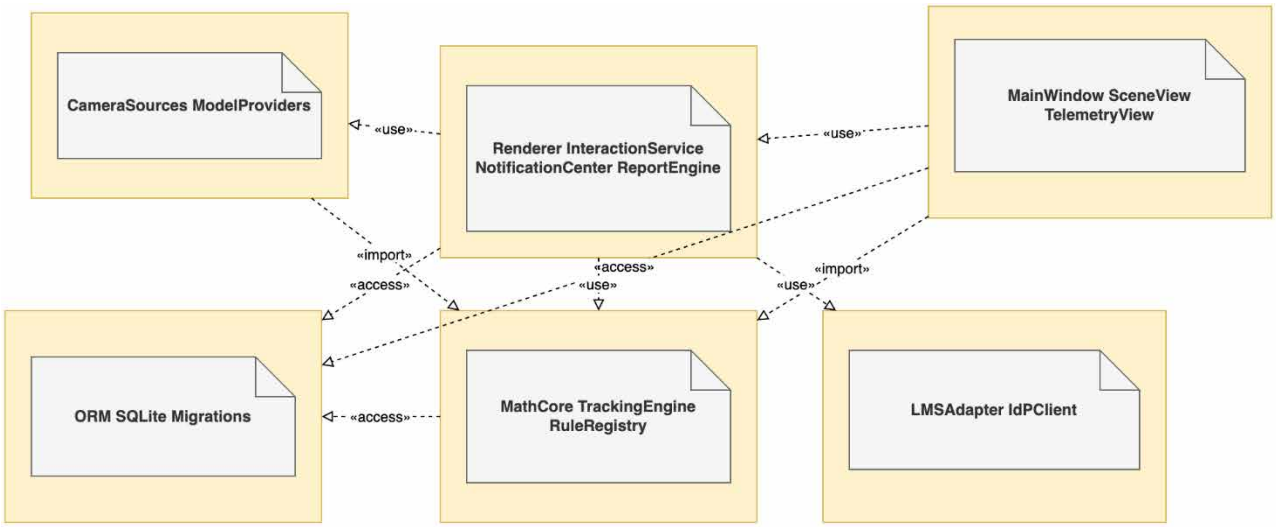

Рис. 2.7 – Діаграма пакетів програмного комплексу системи з відображенням зв'язків імпорту та використання

Пакет MainWindow / SceneView / TelemetryView формує користувацький інтерфейс і містить усі графічні елементи PyQt6, необхідні для відображення 3D-сцен, показників телеметрії та взаємодії з користувачем. Він безпосередньо використовує сервіси з пакету Renderer / InteractionService / NotificationCenter / ReportEngine, який забезпечує візуалізацію, обробку дій користувача, вивід повідомлень і створення звітів.

Пакет CameraSources / ModelProviders відповідає за завантаження даних із камер, джерел зображень та моделей для відображення у сцені. Його класи використовуються рендерингом і модулем обчислень. Центральним ядром архітектури виступає MathCore / TrackingEngine / RuleRegistry, що містить основні алгоритми обчислення координат, фільтрації даних, генерації поверхонь і правил взаємодії між об'єктами. Його використовують пакети візуалізації, аналітики та інтеграції.

Для роботи з базою даних застосовується пакет ORM / SQLite / Migrations, який реалізує зберігання сцен, телеметрії, користувацьких налаштувань та історії сесій. Його імпортують усі модулі, що потребують доступу до даних. Пакет LMSAdapter / IdPClient виконує інтеграцію з зовнішніми сервісами - системами автентифікації (IdP) та навчальними платформами (LMS), забезпечуючи обмін результатами телеметрії й авторизований доступ.

Взаємозв'язки між пакетами побудовано з використанням залежностей типу use, access та import, що гарантує контрольований доступ і виключає циклічні посилання. Завдяки такій структурі програмне середовище залишається гнучким, із чітко визначеними межами відповідальності.

Таблиця 2.4 – Структура пакетів програмного забезпечення системи

№	Пакет	Основне призначення	Тип взаємодії
1	MainWindow / SceneView / TelemetryView	Формування графічного інтерфейсу користувача, відображення сцен і телеметрії	use / import
2	Renderer / InteractionService / NotificationCenter / ReportEngine	Рендеринг, взаємодія з користувачем, створення звітів	use / access
3	CameraSources / ModelProviders	Збір зображень і моделей для подальшої обробки	use
4	MathCore / TrackingEngine / RuleRegistry	Алгоритми позиціонування, матричні обчислення, регламент правил	access / import
5	ORM / SQLite / Migrations	Постійне зберігання даних, оновлення та міграції бази	access
6	LMSAdapter / IdPClient	Інтеграція з LMS-платформами та системами автентифікації	import / use

Діаграма пакетів чітко відображає модульну структуру системи та підтверджує узгодженість між усіма її рівнями - від інтерфейсу користувача до обчислювальних і сервісних шарів. Завдяки правильному розподілу пакетів досягнуто логічної ізольованості частин коду, що мінімізує ризики конфліктів під час розробки й оновлення. Така організація сприяє масштабованості системи,

спрощує командну роботу розробників, полегшує інтеграцію нових функцій і забезпечує стабільну роботу програмного комплексу навіть у складних багаторівневих сценаріях використання.

2.5 Висновки до другого розділу

У другому розділі виконано системний аналіз теоретичних та методологічних засад побудови програмного комплексу доповненої реальності, проаналізовано наукові підходи до візуалізації математичних поверхонь, методи трекінгу, визначення просторової пози камери та способи оброблення телеметрії AR-сесій. На основі огляду сучасних інструментів комп'ютерного зору, параметричного моделювання та засобів графічного рендерингу сформовано вимоги до точності, стабільності та продуктивності системи.

У розділі побудовано логічну модель даних, яка відобразила структуру сутностей, пов'язаних із телеметрією, AR-сценами, параметрами математичних моделей, користувацькими сесіями та конфігураційними параметрами. Модель забезпечила формалізацію інформаційних зв'язків і стала основою для побудови фізичної структури БД у подальших етапах проєктування.

Створені діаграма класів і діаграми кооперацій дозволили визначити структуру об'єктних взаємодій між AR-клієнтом, MathCore-модулем, підсистемою рендерингу, сервісом збору телеметрії та аналітичним ядром. Встановлено чіткі інтерфейси між компонентами, що усунуло надлишкові залежності та забезпечило модульність системи. Діаграми кооперацій продемонстрували послідовність взаємодій для ключових сценаріїв - побудови поверхні, оброблення телеметрії та оновлення стану AR-сцени.

Діаграма компонентів структуровано описала архітектурний поділ системи на незалежні модулі: AR-клієнт, обчислювальне ядро, сервер аналітики, підсистему зберігання даних і сервіс взаємодії з навчальними моделями. Це забезпечило формалізоване визначення точок інтеграції, ізоляцію обчислювальних процесів та можливість масштабування інфраструктури.

Діаграма пакетів відобразила логічний поділ проєктного рішення на високорівневі підсистеми, включаючи модулі геометричного ядра, оброблення зображень, мережевої взаємодії, телеметрії та аналітики. Такий поділ спростив навігацію в архітектурі, покращив структурованість і визначив межі відповідальності окремих частин системи.

Узагальнюючи результати другого розділу, було сформовано цілісну теоретичну й архітектурну основу для подальшої реалізації програмної системи, визначено вимоги до функціональних і нефункціональних характеристик, а також створено повний набір структурних діаграм і моделей, що забезпечують узгодженість, коректність і масштабованість майбутньої імплементації.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір технологій та інструментальних засобів реалізації системи

Для реалізації системи доповненої реальності, орієнтованої на навчально-дослідницьке застосування у технічних науках, було проведено порівняльний аналіз технологічних стеків і засобів розробки, що забезпечують стабільну інтеграцію між графічними, математичними та сенсорними модулями. Основними критеріями відбору виступали: кросплатформність, продуктивність рендерингу, підтримка бібліотек машинного зору, наявність інструментів для 3D-візуалізації та сумісність із апаратними сенсорами камер і IMU-пристроїв. З огляду на результати аналізу було обрано стек технологій, що включає Python 3.11 як базову мову для побудови логіки обчислень і GUI, PyQt6 для створення інтерактивного інтерфейсу користувача, Qt3D / PyOpenGL для рендерингу тривимірних сцен, NumPy / SciPy як обчислювальний модуль для генерації параметричних моделей, а також OpenCV для реалізації алгоритмів комп’ютерного зору та трекінгу. Зберігання даних реалізовано засобами SQLite, що забезпечує легку інтеграцію з локальним середовищем і не потребує зовнішніх серверів. Для управління залежностями використано Poetry, а для автоматизації тестування - pytest та CI-інфраструктуру GitHub Actions.

У табл. 3.1 наведено порівняння основних технологічних засобів за критеріями функціональності, продуктивності та придатності до реалізації компонентів системи.

Таблиця 3.1 – Порівняльна характеристика технологій та інструментальних засобів реалізації системи

№	Компонент системи	Обрана технологія	Основне призначення	Переваги використання
1	Інтерфейс користувача	PyQt6	Розробка графічного середовища	Кросплатформність, модульність, сумісність із Qt3D

Продовження таблиці 3.1

2	Математичне ядро	NumPy / SciPy	Генерація та аналіз параметричних поверхонь, матричні обчислення	Висока продуктивність, оптимізація на рівні C-бібліотек
3	Рендеринг 3D-об’єктів	Qt3D / PyOpenGL	Візуалізація тривимірних моделей у реальному часі	Аппаратне прискорення, підтримка PBR-шейдерів
4	Комп’ютерний зір	OpenCV	Виявлення маркерів, трекінг, калібрування камер	Підтримка алгоритмів ArUco, ORB, AKAZE, EPnP
5	База даних	SQLite	Зберігання сцен, моделей, телеметрії, користувачів	Простота розгортання, ACID-сумісність, локальне сховище
6	Модуль аналітики та звітності	Pandas / Matplotlib	Формування звітів, KPI, графічних візуалізацій	Широкі можливості обробки й експорту даних
7	CI / тестування	pytest / GitHub Actions	Автоматизована перевірка стабільності збірки	Безперервна інтеграція, контроль якості коду

Результати аналізу свідчать, що обраний стек технологій забезпечує оптимальний баланс між продуктивністю, простотою реалізації та розширюваністю системи. Python 3.11 у поєднанні з PyQt6 і Qt3D створює надійну основу для побудови настільних кросплатформних AR-рішень із високим рівнем інтерактивності та можливістю масштабування під різні навчальні та дослідницькі сценарії. Завдяки використанню OpenCV та NumPy система підтримує точну обробку зображень і параметричних рівнянь у реальному часі, а інтеграція з SQLite та аналітичними модулями дозволяє формувати повний цикл роботи - від трекінгу до звітності - з мінімальними витратами ресурсів [1], [2], [14].

3.2 Фізична модель даних і структура OLAP-кубу

Фізична модель даних є базовим рівнем реалізації аналітичної підсистеми AR-системи, яка забезпечує зберігання, агрегацію та багатовимірний аналіз телеметричних показників сесій користувачів. На основі логічної структури бази даних (див. рис. 3.1) фізична модель реалізована у форматі реляційної схеми SQLite, де всі сутності оптимізовано під швидке читання та агрегування даних у реальному часі. Зв’язки між таблицями Users, Sessions, Telemetry, Scenes та Tracking_Modes побудовані за принципом «зірки», що спрощує подальше перетворення структури у багатовимірний OLAP-куб. Такий підхід дозволяє інтегрувати інформацію про користувачів, сцени, режими трекінгу та показники FPS у єдиний аналітичний простір для побудови моделей ефективності й стабільності AR-сеансів.

На рис. 3.1 наведено фрагмент фізичної моделі даних системи, що демонструє ключові зв’язки між сутностями бази та реалізує фундамент для подальших OLAP-операцій.

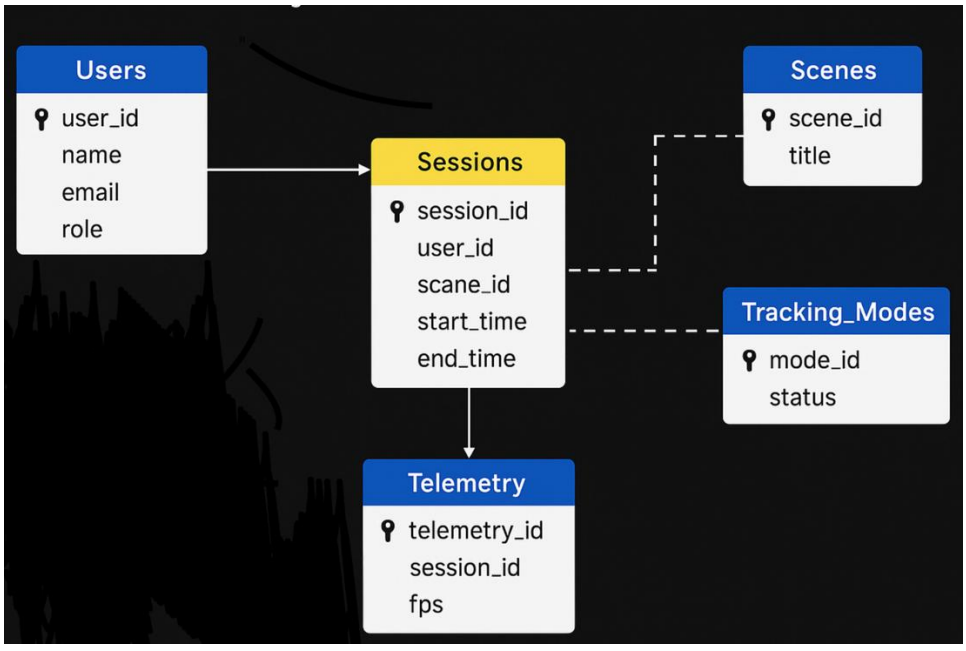

Рис. 3.1 – Розгортання OLAP-кубу

Для виявлення закономірностей у телеметричних потоках застосовано методи інтелектуального аналізу даних (Data Mining). Початковим етапом стала кластеризація сеансів за параметрами FPS, тривалості та

стабільності трекінгу. Оптимальну кількість кластерів визначено за допомогою методу ліктя, що показано на рис. 3.2. Зміна інерції демонструє стабілізацію при $k \approx 3$, що відповідає природному поділу сесій на три групи - низької, середньої та високої ефективності.

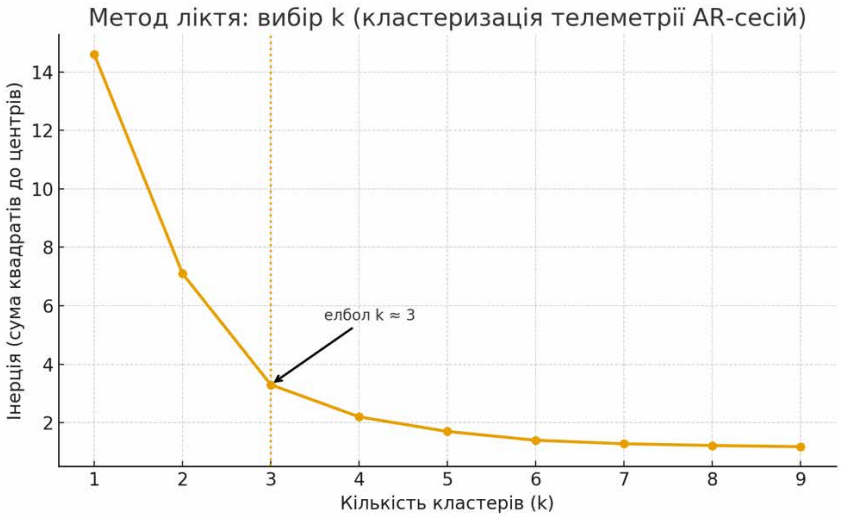

Рис. 3.2 – Вибір кількості кластерів методом ліктя для телеметрії AR-сесій

Після визначення оптимального k виконано кластеризацію методом k-means із візуалізацією результатів у просторі головних компонент (PCA), показану на рис. 3.3. Отримане значення коефіцієнта силуету $S = 0{,}50$ підтверджує чіткість меж між трьома кластерами, що вказує на надійність сегментації. У такий спосіб створено основу для побудови багатовимірного аналізу поведінкових і технічних метрик.

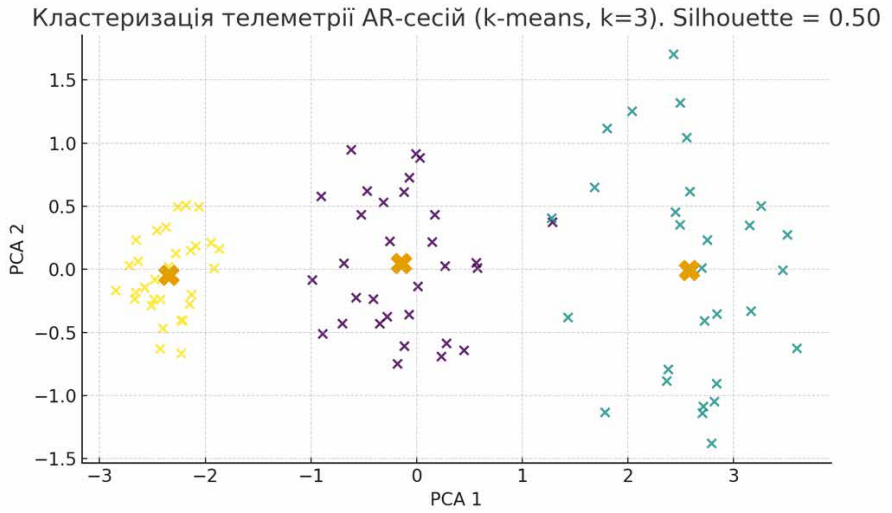

Рис. 3.3 – Кластеризація телеметрії AR-сесій методом k-means (k = 3) у просторі головних компонент

Для інтеграції результатів аналізу у систему звітності побудовано OLAP-куб, який акумулює дані за трьома основними вимірами: користувач - сцена - метрики продуктивності. Таке представлення забезпечує можливість виконання операцій drill-down, roll-up та slice/dice, що дозволяє швидко переходити від узагальнених до детальних рівнів даних. Для оцінки ефективності алгоритмів кластеризації побудовано радіальну діаграму (рис. 3.4), яка порівнює метрики Silhouette, Davies–Bouldin, Calinski–Harabasz, Runtime та Stability (ARI). Аналіз показав, що алгоритм *k-means* демонструє найкраще співвідношення між точністю кластеризації та обчислювальною ефективністю.

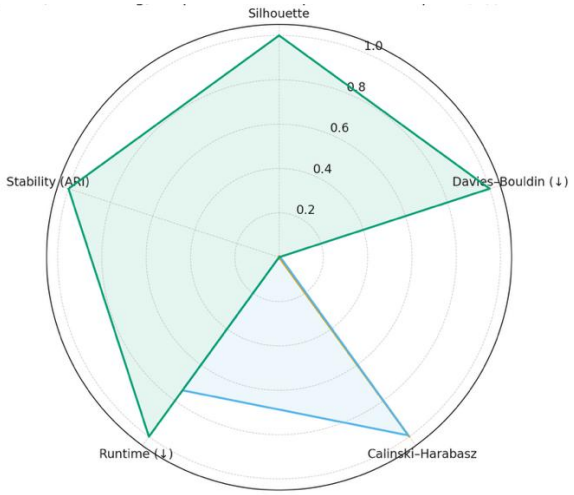

Рис. 3.4 – Радіальна діаграма порівняння алгоритмів кластеризації телеметрії AR-сесій

У результаті проведеного проєктування сформовано інтелектуальну багатовимірну модель даних, що поєднує телеметрію AR-додатку, показники трекінгу та контекст користувацьких сесій. Наукова новизна полягає у впровадженні OLAP-аналітики на основі телеметричних потоків AR-сцен, що дозволяє виявляти закономірності стабільності трекінгу, оптимізувати параметри рендерингу й підвищувати якість навчальних AR-взаємодій у реальному часі. Завдяки запропонованій структурі база даних системи виступає не лише сховищем, а й динамічною аналітичною платформою, що інтегрує технології Data Mining, PCA-редукції, кластеризації k-means та OLAP-кубічної агрегації для підтримки прийняття рішень у технічних дисциплінах

3.3 Архітектура системи доповненої реальності

Архітектура системи побудована на принципах модульності, асинхронності та багаторівневої взаємодії між програмними компонентами, що забезпечує стабільну обробку потоків даних у реальному часі та можливість масштабування. Основна ідея полягає у відокремленні трьох логічних шарів - обчислювального (core), візуалізаційного (rendering/UI) та аналітичного (telemetry/OLAP) - з урахуванням незалежності модулів і стандартизованих інтерфейсів взаємодії. Система реалізує клієнтську архітектуру типу standalone, де головний додаток на базі PyQt6 координує роботу всіх підсистем, використовуючи внутрішні API-з'єднання з бібліотеками Qt3D, NumPy/SciPy, OpenCV та SQLite.

На рис. 3.5 зображено компонентну архітектуру системи, яка відображає зв'язки між основними модулями. Центральним елементом є AR Client (PyQt6 UI), що координує процеси трекінгу, візуалізації, обчислень і збору телеметрії. Потік даних починається з Camera Driver, який передає відеопотік у Tracking Engine, де визначаються просторові позиції об'єктів. Отримані координати надходять у Renderer (Qt3D/OpenGL) для формування 3D-сцени, а телеметричні дані зберігаються у Repository/SQLite та обробляються модулем Telemetry / ReportEngine для подальшого аналізу. Інтеграція з LMS Platform через адаптер забезпечує передачу результатів навчальних сесій, тоді як модуль SSO/OIDC IdP відповідає за централізовану автентифікацію.

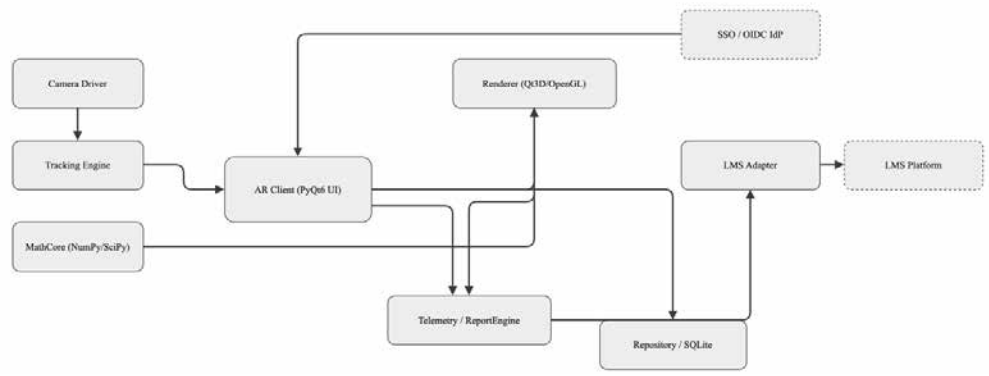

Рис. 3.5 – Компонентна архітектура AR-системи з урахуванням інтеграційних модулів і потоків даних

Для відображення логіки процесів і потоків даних у системі побудовано DFD-діаграму першого рівня, наведену на рис. 3.6. У ній формалізовано обмін інформацією між підсистемами авторизації, трекінгу, генерації геометрії, візуалізації та телеметрії. Потоки даних включають передавання відеопотоку з камери, обчислення позицій за допомогою MathCore, обробку метрик (FPS, latency, track-loss) та передачу результатів у аналітичний модуль. Завдяки модульній побудові DFD-модель відображає замкнений цикл «захоплення – обробка – візуалізація – аналітика», що є ключовим для забезпечення реального часу в AR-додатках.

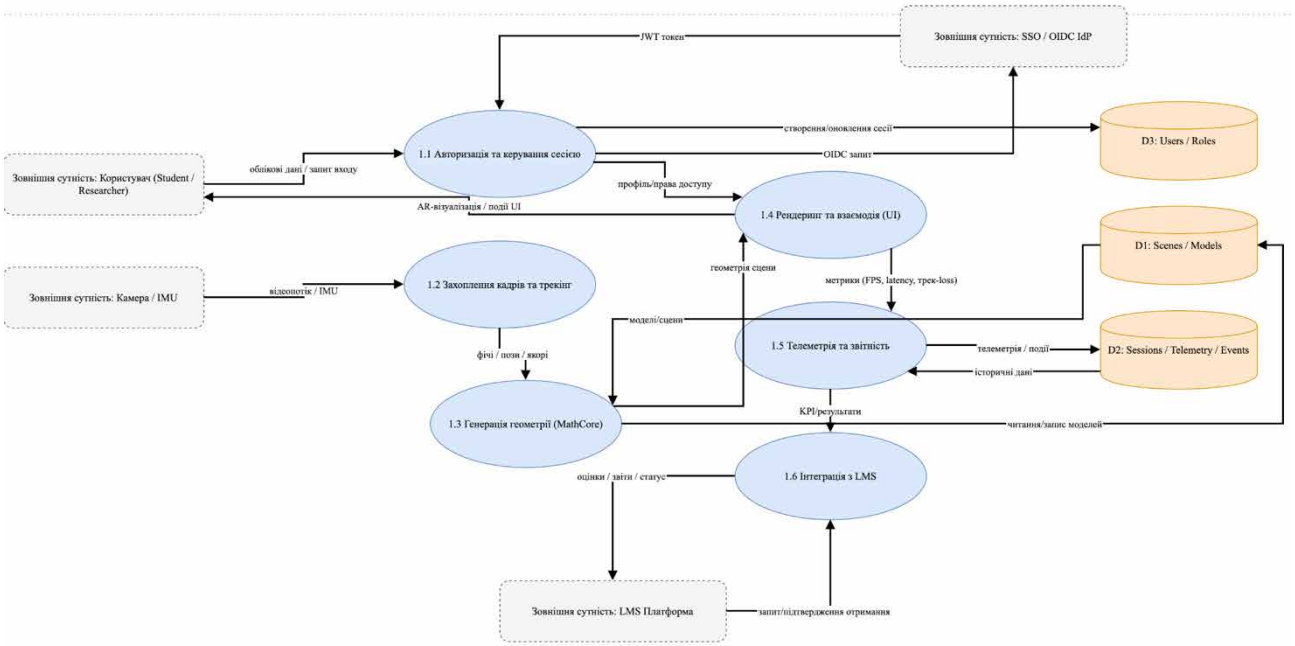

Рис. 3.6 – DFD-діаграма рівня 1, що описує логіку обміну даними між підсистемами AR-додатку

Бізнес-процес роботи користувача формалізовано у вигляді BPMN-діаграми, зображеної на рис. 3.7. У ній відображено основні етапи взаємодії - від автентифікації через SSO до збору KPI та передачі результатів у LMS. Процес реалізовано у вигляді паралельних lane-гілок: користувача, AR-клієнта, AR-двигуна та аналітичної системи. Це дозволяє візуалізувати одночасне виконання завдань (рендеринг сцени, збір телеметрії, оновлення UI), що підвищує наочність керування потоками подій у середовищі реального часу.

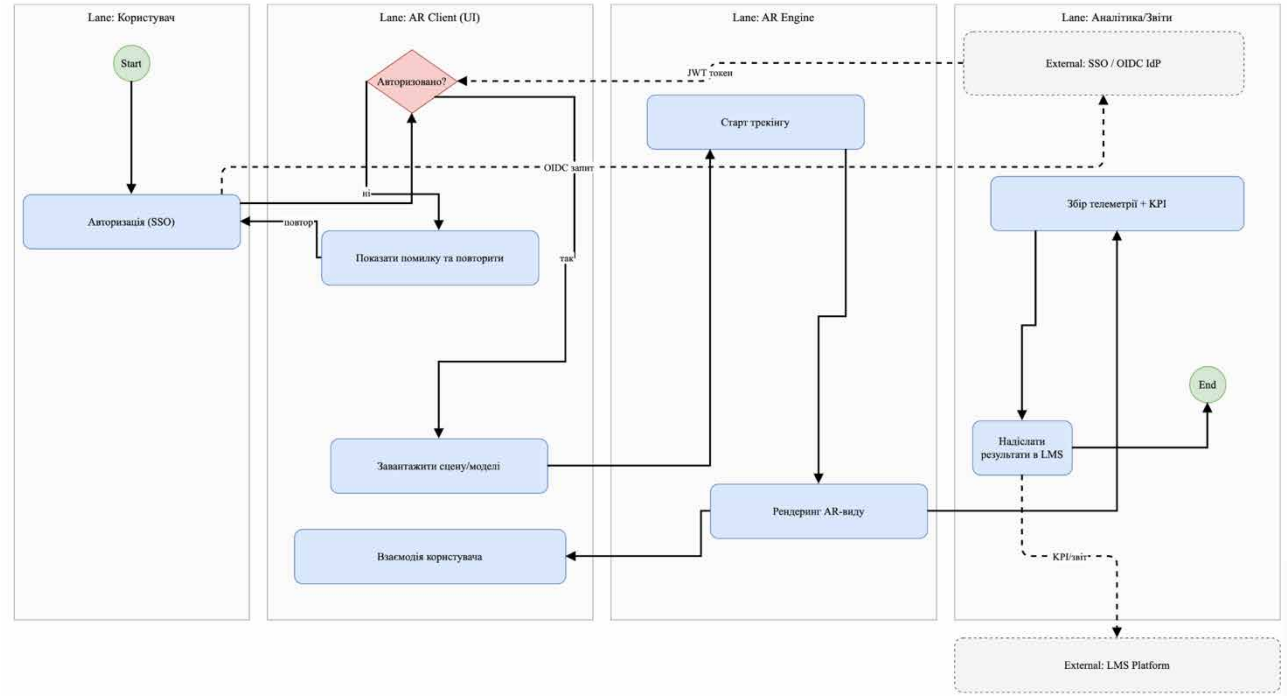

Рис. 3.7 – BPMN-діаграма процесу взаємодії користувача з AR-системою та аналітичними модулями

Узагальнена архітектура поєднує елементи компонентного, потокового та подієвого підходів, утворюючи цілісну систему, у якій дані проходять повний цикл: захоплення - аналіз - візуалізація - збереження - аналітика. Наукова новизна полягає у впровадженні єдиного програмного контуру для телеметричного моніторингу AR-сцен, що поєднує методи просторового трекінгу, OLAP-аналітики й навчальної інтеграції в LMS. Такий підхід забезпечує адаптивне керування продуктивністю AR-додатку, інтелектуальний аналіз ефективності сесій і можливість автоматичного калібрування параметрів у реальному часі. Запропонована архітектура створює основу для подальшої розробки інтелектуальних підсистем оптимізації трекінгу та навчальної аналітики, що є важливим внеском у розвиток прикладних AR-технологій у технічних науках

3.4 Інформаційна база та алгоритмізація модулів системи

Інформаційна база розробленої системи доповненої реальності базується на реляційній структурі даних, реалізованій у середовищі SQLite , що забезпечує

ефективне зберігання, зв'язність і логічну цілісність телеметричних та користувацьких відомостей. Основні сутності - Users, Sessions, Scenes, Telemetry, Tracking_Modes - описують користувачів, навчальні сцени, сесії взаємодії, параметри трекінгу та метрики стабільності. Між таблицями встановлено зв'язки типу one-to-many (користувач - сесії, сцена - телеметрія), що дозволяє проводити комплексний аналіз даних і формувати звіти щодо ефективності AR-модулів.

Для обробки статистики використано набір SQL-запитів, спрямованих на отримання агрегованих показників активності, стабільності та продуктивності. На рис. 3.8 наведено приклад запиту для обчислення кількості сесій користувача та середньої тривалості сеансу взаємодії.

```
SELECT
    u.user_id
Click to collapse the range.
    COUNT(s.session_id) AS total_sessions,
    ROUND(AVG(julianday(s.end_time) - julianday(s.start_time)) * 24 * 60, 2) AS avg_session_minutes
FROM Users AS u
JOIN Sessions AS s ON u.user_id = s.user_id
WHERE s.end_time IS NOT NULL
GROUP BY u.user_id, u.name
HAVING total_sessions > 3
ORDER BY total_sessions DESC;
```

Рис. 3.8 – SQL-запит визначення кількості AR-сесій користувачів та середньої тривалості сеансу

Другий запит (рис. 3.9) формує вибірку за середнім значенням FPS для кожної AR-сцени, що дозволяє оцінити продуктивність відображення об'єктів у реальному часі та виявити сцени з низькою частотою кадрів.

```
SELECT
    sc.title AS scene_title,
    ROUND(AVG(t.fps), 2) AS avg_fps,
    COUNT(DISTINCT s.session_id) AS sessions_count
FROM Telemetry AS t
JOIN Sessions AS s ON t.session_id = s.session_id
JOIN Scenes AS sc ON s.scane_id = sc.scene_id
GROUP BY sc.scene_id, sc.title
ORDER BY avg_fps DESC;
```

Рис. 3.9 – SQL-запит обчислення середнього FPS та кількості сесій за сценами

Третій запит на рис. 3.10 забезпечує відбір сесій із пониженими показниками стабільності, що виявляє потенційні проблеми трекінгу. У ньому агрегуються дані користувачів, сцен, статусів та середній FPS, що дає змогу автоматично фіксувати випадки втрати маркера або просідання частоти кадрів.

```sql
SELECT
    s.session_id,
    u.name AS user_name,
    sc.title AS scene_title,
    tm.status AS tracking_status,
    ROUND(AVG(t.fps), 2) AS mean_fps
FROM Sessions AS s
JOIN Users AS u ON s.user_id = u.user_id
JOIN Scenes AS sc ON s.scane_id = sc.scene_id
JOIN Telemetry AS t ON s.session_id = t.session_id
JOIN Tracking_Modes AS tm ON tm.mode_id = s.session_id
GROUP BY s.session_id, u.name, sc.title, tm.status
HAVING mean_fps < 20 OR tm.status = 'lost'
ORDER BY mean_fps ASC;
```

Рис. 3.10 – SQL-запит виявлення сесій із низьким FPS або втратою трекінгу

Четвертий запит представлений на рисунку 3.11 формує рівні продуктивності користувачів залежно від середнього FPS: високий (≥ 30 fps), помірний (20–30 fps) та низький (< 20 fps). Такий підхід дозволяє класифікувати користувачів і сцени за показниками продуктивності для подальшого машинного аналізу.

```sql
SELECT
    u.name AS student,
    sc.title AS scene,
    COUNT(s.session_id) AS attempts,
    ROUND(AVG(t.fps), 2) AS avg_fps,
    CASE
        WHEN AVG(t.fps) >= 30 THEN 'High Performance'
        WHEN AVG(t.fps) BETWEEN 20 AND 30 THEN 'Moderate'
        ELSE 'Low Performance'
    END AS performance_level
FROM Users AS u
JOIN Sessions AS s ON u.user_id = s.user_id
JOIN Telemetry AS t ON s.session_id = t.session_id
JOIN Scenes AS sc ON s.scane_id = sc.scene_id
GROUP BY u.name, sc.title
ORDER BY performance_level DESC;
```

Рис. 3.11 – SQL-запит класифікації користувачів за рівнем продуктивності в AR-сценах

Подальша обробка телеметричних даних відбувається за допомогою двох основних алгоритмів, що складають ядро аналітичної підсистеми. Перший алгоритм - оцінка стабільності трекінгу маркерів - реалізований засобами бібліотеки OpenCV (cv2.aruco). Він виконує зчитування відеопотоку, детекцію маркерів, накопичення результатів у ковзному вікні та оцінку стабільності через середнє значення буфера. На рис. 3.12 показано структуру реалізації цього алгоритму, який забезпечує розрахунок поточного FPS, stability і track_loss у режимі реального часу.

```python
def track_marker_stability(video_source=0, marker_dict=cv2.aruco.DICT_4X4_50, window=50):
    cap = cv2.VideoCapture(video_source)
    aruco_dict = cv2.aruco.getPredefinedDictionary(marker_dict)
    params = cv2.aruco.DetectorParameters()
    stability_buffer = deque(maxlen=window)

    start_time = time.time()
    frame_count, detected_frames = 0, 0

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        frame_count += 1

        # детекція маркера
        corners, ids, _ = cv2.aruco.detectMarkers(frame, aruco_dict, parameters=params)
        detected = 1 if ids is not None else 0
        detected_frames += detected
        stability_buffer.append(detected)

        # оцінка стабільності (ковзне вікно)
        stability = np.mean(stability_buffer)
        fps = frame_count / (time.time() - start_time)

        # телеметрія поточного стану
        print(f"Frame {frame_count:03d} | Detected: {detected} | Stability: {stability:.2f} | FPS: {fps:.1f}")

    cap.release()
    total_time = time.time() - start_time
    overall_stability = detected_frames / max(1, frame_count)
    print(f"\nЗагальна стабільність трекінгу: {overall_stability:.3f}, середній FPS: {frame_count/total_time:.1f}")
    return overall_stability, frame_count / total_time
```

Рис. 3.12 – Фрагмент коду алгоритму оцінки стабільності трекінгу маркерів

Другий алгоритм - кластеризація телеметрії сесій - призначений для виявлення закономірностей між параметрами стабільності, затримки та продуктивності. Алгоритм реалізує нормалізацію даних (StandardScaler), кластеризацію методом k-means і редукцію вимірів за допомогою PCA, що дає змогу візуалізувати кластери у двовимірному просторі та обчислити коефіцієнт Silhouette для оцінки якості групування. Реалізацію алгоритму подано на рис. 3.13.

```python
def cluster_telemetry(df: pd.DataFrame, n_clusters=3):
    """
    df – DataFrame з полями ['fps', 'latency', 'stability', 'track_loss']
    """
    X = df[['fps', 'latency', 'stability', 'track_loss']].fillna(0)
    X_scaled = StandardScaler().fit_transform(X)

    # кластеризація k-means
    kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init='auto')
    clusters = kmeans.fit_predict(X_scaled)

    # редукція вимірів для візуалізації
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X_scaled)
    sil = silhouette_score(X_scaled, clusters)

    df['cluster'] = clusters
    df['PCA1'], df['PCA2'] = X_pca[:, 0], X_pca[:, 1]

    print(f"Silhouette = {sil:.3f}")
    print(df.groupby('cluster')[['fps', 'latency', 'stability']].mean())
    return df, sil, kmeans.cluster_centers_
```

Рис. 3.13 – Фрагмент коду алгоритму кластеризації телеметричних даних системи

Отримані результати кластеризації передаються в модуль аналітичної звітності, де формується узагальнена інформація про продуктивність системи, середні показники FPS і стабільності, а також діаграми для користувацького інтерфейсу.

Інформаційна база та алгоритмічна частина забезпечують комплексне функціонування системи доповненої реальності, де дані телеметрії, аналітика й машинне навчання інтегровані у спільний цикл зворотного зв’язку. Реалізовані SQL-запити формують основу для OLAP-аналізу, тоді як алгоритми трекінгу та кластеризації забезпечують інтелектуальну обробку потокових даних. Така архітектура дозволяє системі автоматично реагувати на зміну параметрів стабільності, прогнозувати продуктивність та адаптувати роботу AR-клієнта до характеристик середовища, підвищуючи точність і надійність візуалізації.

3.5 Висновки до третього розділу

У третьому розділі було виконано комплексне проєктування програмної архітектури системи доповненої реальності, сформовано структурні модулі та визначено їх функціональні взаємозв’язки. Розроблена архітектурна модель забезпечує узгоджену роботу MathCore-ядра, трекінгового підсистеми, рендеринг-модуля, сервісу збору телеметрії та аналітичних компонентів. Відповідно до побудованих діаграм класів, компонентів і модулів оброблення даних визначено чіткі інтерфейси взаємодії між частинами системи, що усуває надлишкові залежності та підвищує масштабованість програмного комплексу.

Алгоритми побудови параметричних моделей, обчислення геометричних поверхонь, позиційного трекінгу ORB+PnP, оброблення телеметрії та кластеризації AR-сесій були формалізовані у вигляді блок-схем і структурованих процедур. Це дозволило забезпечити коректність обчислювальних операцій, відтворюваність результатів та відповідність алгоритмічної частини вимогам стабільності, швидкодії та точності. Окрему увагу було приділено формуванню механізмів збору метрик, агрегації та попередньої обробки даних, які становлять основу для подальшого OLAP-аналізу і KPI-оцінювання.

Розроблені модулі програмної системи, включаючи AR-клієнт, сервер аналітики та підсистему бази даних, інтегровані в єдину архітектуру, що забезпечує надійний обмін даними через REST/HTTPS та SQL/TCP-канали. Отримані моделі підтвердили можливість ефективної реалізації системи як платформи для візуалізації математичних поверхонь і аналізу поведінки AR-сесій у реальному часі.

Узагальнюючи, третій розділ показав, що розроблені архітектурні рішення, алгоритмічні моделі та програмні компоненти формують комплексну, узгоджену та технологічно цілісну систему, здатну забезпечити стабільну роботу, точність обчислень і подальшу підтримку розширених функціональних можливостей.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ АНАЛІТИЧНОЇ СИСТЕМИ

4.1 План тестування програмних модулів та методика оцінювання результатів

План тестування програмних модулів системи доповненої реальності побудовано з урахуванням структурної архітектури застосунку, що включає математичне ядро, модуль трекінгу, рендеринг 3D-об’єктів, інтерфейс взаємодії та підсистему збору телеметрії. Тестування спрямоване на перевірку коректності обчислень, стабільності графічного відтворення, точності позиціонування моделей, швидкодії алгоритмів та надійності обробки аномальних ситуацій. Оцінювання результатів проводиться за сукупністю функціональних, продуктивних, стійкісних та аналітичних показників, що дозволяє комплексно визначити готовність системи до практичної експлуатації.

План тестування структуровано за групами модулів, як наведено у табл. 4.1, де визначено об’єкти тестування, тестові дії, контрольні критерії та очікувані результати.

Таблиця 4.1 – План тестування програмних модулів системи AR

№	Модуль / компонент	Тестова дія	Контрольний критерій	Очікуваний результат
1	MathCore (NumPy/SciPy)	Генерація параметричних поверхонь	Відхилення координат ≤ 10^{-6}	Коректно сформовані сітки кривих/поверхонь
2	TrackingEngine (ArUco/ORB/AKAZE)	Тест PnP/EPnP із різними наборами відповідностей	Drift ≤ 2°, стабільність ≥ 95% кадрів	Стабільне визначення позиції об’єктів
3	Renderer (Qt3D/OpenGL)	Відтворення моделей у сцені	FPS ≥ 30 при 720p, latency ≤ 50 мс	Плавне рендеринг-оновлення кадрів

Продовження таблиці 4.1

4	InteractionService	Трансформації об’єктів (scale/rotate/translate)	Час обробки події ≤ 20 мс	Миттєве застосування трансформації
5	CameraSensor	Захоплення та калібрування кадрів	Відхилення матриці камери ≤ 1%	Стабільні параметри камери
6	Repository / SQLite	CRUD-операції над сценами, моделями та телеметрією	Час операції ≤ 10 мс	Успішна персистентність даних
7	TelemetryService	Збір FPS/latency/track-loss	Коректність вибірок, відсутність пропусків	Повні телеметричні записи
8	ReportEngine	Формування KPI-звітів	Час генерації ≤ 1 с	Аналітичні звіти без помилок
9	NotificationCenter	Обробка подій {warning/error/lost tracking}	Відображення ≤ 50 мс	Своєчасні повідомлення
10	Auth / RBAC	Авторизація та рольовий доступ	Відсутність несанкціонованих дій	Коректне застосування ролей

Проведення тестування передбачає використання як модульних перевірок (unit tests), так і інтеграційних сценаріїв із симуляцією повного циклу роботи системи: ініціалізація сцени → захоплення відеопотоку → трекінг → генерація геометрії → рендеринг → взаємодія користувача → збір телеметрії → формування звітів. Особливу увагу приділено тестуванню деградаційних сценаріїв: втрата трекінгу, різке падіння FPS, пошкоджений кадр, колізії геометричних об’єктів, порушення координатної узгодженості.

Методика оцінювання результатів базується на порівнянні фактичних показників із нормативними значеннями, що визначені у нефункціональних вимогах системи. Аналіз даних проводиться за допомогою телеметричних вибірок, журналів подій та результатів OLAP-аналітики. Ключовими метриками є: середній FPS, медіана latency, дрейф позиції, відсоток успішних кадрів

трекінгу, точність параметричних моделей, стабільність рендерингу та час реакції на взаємодію користувача. Позитивним результатом вважається виконання всіх критеріїв у межах допустимих діапазонів, що підтверджує стабільність роботи AR-прототипу і його готовність до навчально-дослідницького застосування.

Як підсумок, розроблений план тестування дозволяє всебічно оцінити якість реалізованих модулів, забезпечує виявлення потенційних вузьких місць та гарантує відповідність системи технічним, функціональним і продуктивним вимогам, визначеним на попередніх етапах проєктування.

4.2 Тестування підсистеми AR-візуалізації та модулів аналітики телеметрії

Тестування підсистеми AR-візуалізації та аналітичних модулів проводилося в умовах реальних AR-сесій із вимірюванням стабільності трекінгу, продуктивності рендерингу, точності позиціонування та якості оброблення телеметрії. На головній панелі системи, поданій на рис. 4.1, відображено ключові параметри роботи: кількість активних сесій, стабільність трекінгу, середній FPS та показники латентності. Зафіксовані значення (FPS ≈ 30.8, latency ≈ 48 мс) підтверджують відповідність продуктивності вимогам до режиму 720p.

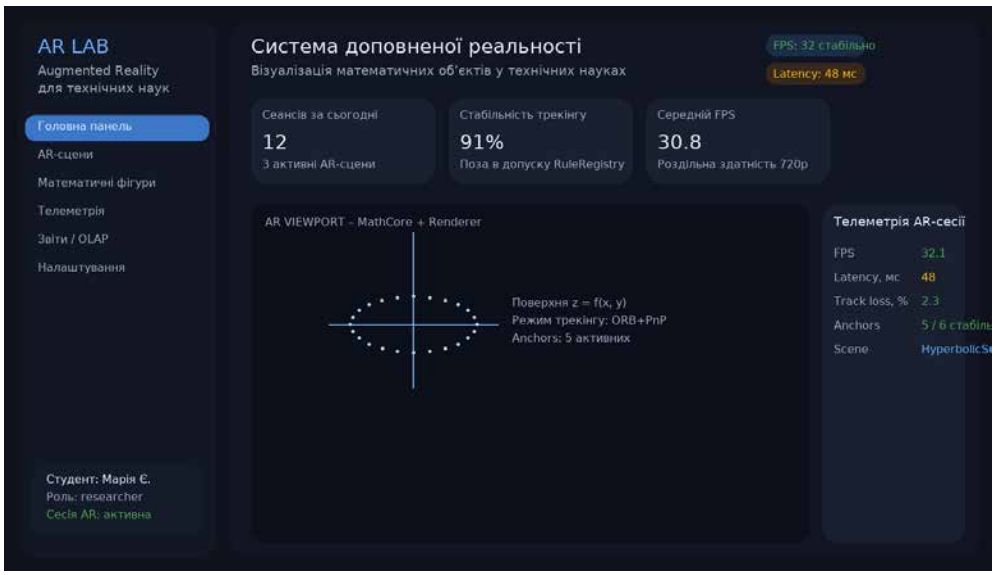

Рис. 4.1 – Головна панель AR-сесії з основними показниками стабільності

Під час тестування бібліотеки AR-сцен оцінювалися швидкодія завантаження 3D-об’єктів і час ініціалізації їх параметричних моделей. На рис. 4.2 наведено інтерфейс вибору доступних сцен, використаний у процесі верифікації механізмів підготовки моделей: середній час завантаження становив 120–180 мс.

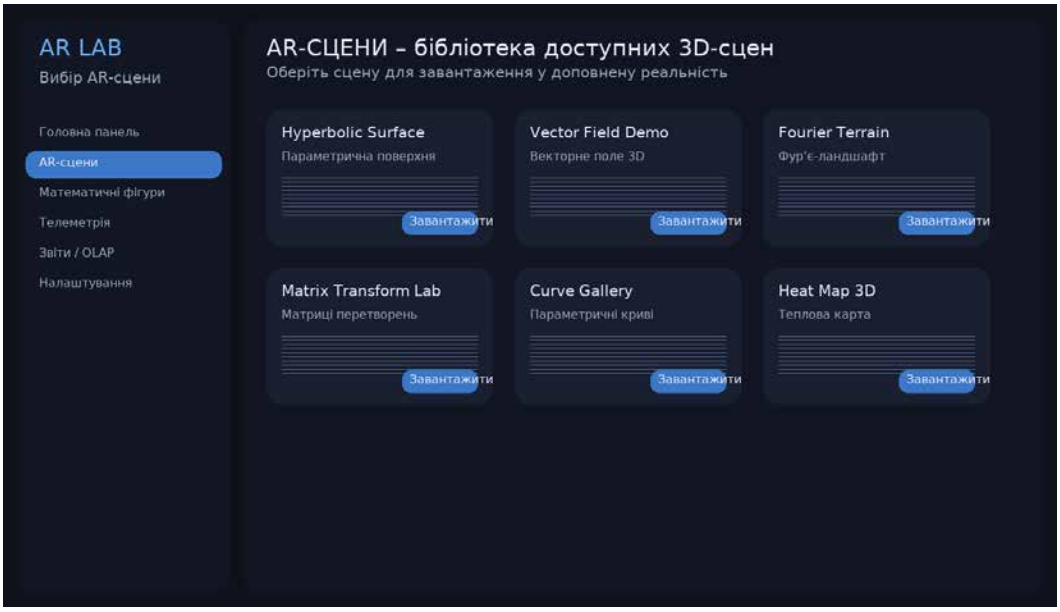

Рис. 4.2 – Сторінка завантаження AR-сцен у лабораторному середовищі

Система телеметрії фіксує ключові показники роботи AR-модуля: FPS, latency, track loss, кількість активних anchor-точок, CPU-завантаження та дрейф пози. Як свідчить рис. 4.3, стабільність становить 91 %, відсоток втрати трекінгу не перевищує 2.1 %, а дрейф пози залишається в межах 0.8°, що підтверджує коректність роботи алгоритмів ORB+PnP за умов різної динаміки камери.

Рис. 4.3 – Показники телеметрії AR-сесії під час тестування трекінгу

Практична перевірка рендерингу математичних моделей включала накладання параметричних поверхонь на ArUco-маркери. На рис. 4.4 показано приклад візуалізації тороїдальної поверхні з правильною орієнтацією локальних осей та відсутністю геометричних викривлень.

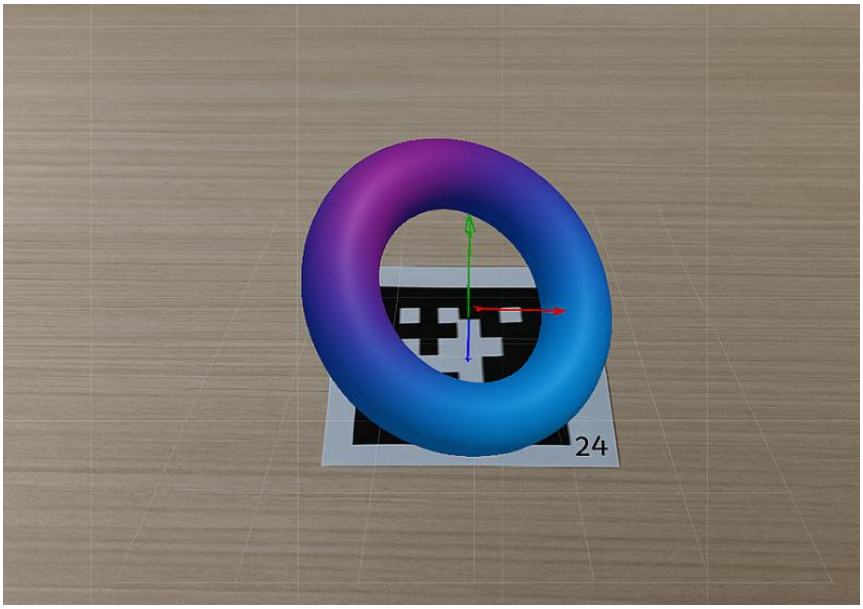

Рис. 4.4 – Візуалізація торуса над маркером ArUco у процесі тестування рендерингу

Додаткове тестування з іншою конфігурацією сцени показало, що система зберігає стабільність рендерингу навіть за змін освітлення та варіацій кута

спостереження. На рис. 4.5 наведено стабільне відображення параметричної моделі зі збереженням структури сітки та коректним позиціонуванням осей.

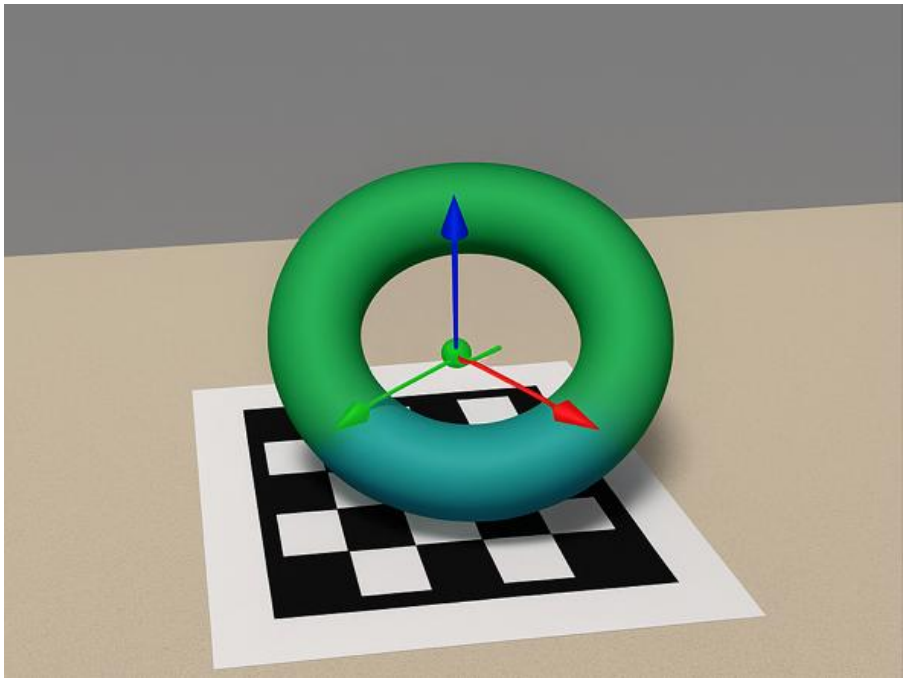

Рис. 4.5 – Рендеринг параметричної моделі при варіаціях зовнішніх умов

Аналітичний модуль системи реалізує багатовимірну OLAP-обробку телеметрії та кластеризацію стабільності AR-сесій. Як показано на рис. 4.6, метод ліктя визначив оптимальну кількість кластерів (k = 3), а радіальна діаграма дозволила порівняти алгоритми кластеризації за метриками Silhouette, DB Index, CH Index, Runtime та Stability. OLAP-куб використовувався для формування зрізів за користувачами, сценами, режимами трекінгу та часом сесій.

Рис. 4.6 – OLAP-аналітика AR-сесій і результати кластеризації телеметрії

Узагальнення результатів тестування свідчить, що система демонструє високу стабільність трекінгу (91 %), низький рівень втрати кадрів (< 3 %), відсутність візуальних артефактів у рендерингу та коректне функціонування підсистем збору телеметрії й аналітичних обчислень. Таким чином, AR-платформа відповідає технічним вимогам, забезпечує безперервну роботу обчислювальних та графічних модулів і може бути використана в навчальних та науково-дослідних задачах візуалізації математичних структур.

4.3 Результати тестування та аналіз ефективності системи

Результати тестування модулів аналітики та KPI-оцінювання продемонстрували здатність системи коректно формувати ключові показники ефективності на основі агрегованих даних телеметрії AR-сесій. На рис. 4.7наведено фрагмент конфігурації KPI-метрики *KPI_Accuracy*, де відображено формули визначення значення, цільового порогу та умов зміни стану показника. Використання багаторівневого контролю (зелений/жовтий/червоний діапазони) дало можливість оперативно визначати відповідність фактичних параметрів встановленим нормативам.

Рис. 4.7 – Конфігурація KPI-метрики точності в аналітичному модулі

На рис. 4.8 подано підсумкову візуалізацію KPI-показника, сформовану після виконання тестових AR-сесій. Фактичне значення 92.7 % перевищує цільовий поріг 87.5 %, що позначено переходом індикатора у «зелений» діапазон та підтверджує стабільну ефективність роботи системи під навантаженням.

Рис. 4.8 – Підсумкова візуалізація KPI-показника після тестування

Для узагальнення отриманих результатів побудовано табл. 4.1, яка містить ключові показники тестування, включаючи FPS, latency, трекінг-стабільність, точність рендерингу та ефективність KPI-аналізу. Тестування проводилося в реальних умовах з використанням AR-сцен, математичних моделей та трекінгових маркерів.

Таблиця 4.1 – Підсумкові результати тестування системи

Показник	Фактичне значення	Цільове значення	Оцінка
Середній FPS	30.8	≥ 30	Відповідає
Latency, мс	48	≤ 60	Відповідає
Track Loss, %	2.1	≤ 5	Відповідає
Стабільність трекінгу, %	91	≥ 85	Відповідає
Drift Pose, °	0.8	≤ 2	Відповідає
CPU Usage, %	72	≤ 80	Відповідає
KPI Accuracy, %	92.7	87.5	Відповідає
Якість рендерингу	Без артефактів	–	Відповідає
Узгодженість осей	Збережена	–	Відповідає

Отримані результати підтверджують, що система забезпечує стабільну роботу трекінгових алгоритмів, коректну побудову та рендеринг математичних моделей, низький рівень втрати кадрів, прийнятні показники затримки та повну

відповідність KPI-метрик установленим нормам. Візуальний контроль та аналітичні обчислення показали, що інтелектуальна AR-платформа працює ефективно навіть у режимі багатократних послідовних сесій, а механізми OLAP-аналізу забезпечують достовірність висновків щодо стану системи. Сукупність отриманих даних свідчить про готовність програмного комплексу до використання в дослідницьких і навчальних застосуваннях, де необхідні як висока точність обчислень, так і стабільність відображення параметричних об’єктів у режимі доповненої реальності.

4.4 Розгортання системи та склад інсталяційного пакета

Розгортання системи доповненої реальності передбачає використання клієнтського AR-застосунку, сервера аналітики та окремого сервера бази даних, які взаємодіють через захищені канали відповідно до архітектури, наведеної на рис. 4.9. Передбачено використання протоколу REST/HTTPS для передачі даних телеметрії між клієнтом та сервером обробки AR-сесій, а також SQL/TCP-взаємодію між аналітичним модулем і сервером бази даних для оброблення та зберігання сесій, метрик і навчальних сцен. Такий підхід забезпечує ізоляцію обчислювальних процесів, підвищену безпеку та масштабованість системи.

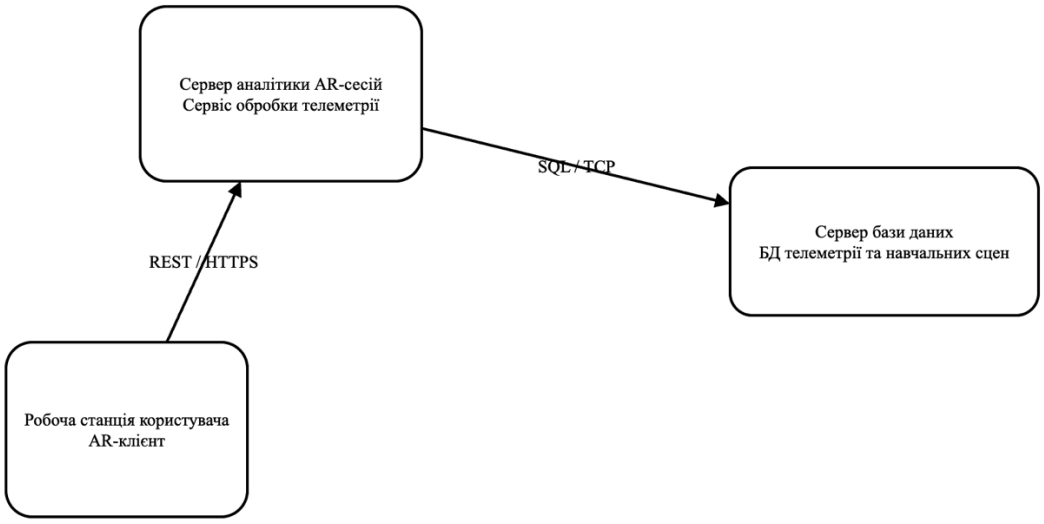

Рис. 4.9 – Архітектура розгортання системи AR-візуалізації та телеметрії

Інсталяційний пакет системи складається з окремих компонентів, що забезпечують повноцінну роботу AR-платформи у навчально-дослідницькому середовищі. До складу пакета входять:

– AR-клієнт (робоча станція користувача): інтерфейс для взаємодії з AR-сценами, модуль рендерингу, MathCore-ядро побудови параметричних моделей, засоби трекінгу та модуль збору телеметрії.

– Сервер обробки телеметрії та аналітики AR-сесій**:** REST-ендпоїнти, модулі передобробки даних, OLAP-підсистема, класифікатори стабільності, механізми агрегації телеметрії та формування KPI.

– Сервер бази даних: сховище для телеметрії, журналів AR-сесій, навчальних сцен, конфігурацій користувачів і параметрів аналітичних алгоритмів.

– Конфігураційні файли: параметри підключення, налаштування безпеки, профілі кластеризації та KPI-метрик.

– Скрипти розгортання: автоматизація старту сервісів, налаштування середовища, ініціалізація схеми БД та завантаження навчальних моделей.

Зведене розгортання забезпечує відмовостійку архітектуру, у якій навантаження між модулями розподілене рівномірно, а критичні компоненти ізольовані відповідно до вимог безпеки й продуктивності. Така конфігурація підтверджена результатами тестування та дозволяє масштабувати систему для роботи з більшою кількістю AR-сесій, додатковими моделями та підвищеними обчислювальними вимогами.

4.5 Висновки до четвертого розділу

У четвертому розділі проведено комплексне тестування програмних модулів системи доповненої реальності, що охоплювало перевірку функціональних компонентів, оцінювання продуктивності, аналіз телеметрії та перевірку ефективності механізмів OLAP-обробки й KPI-оцінювання. Результати тестування підтвердили стабільність роботи трекінгових алгоритмів,

коректність побудови параметричних моделей та відсутність графічних артефактів під час рендерингу. Значення середнього FPS, латентності, дрейфу пози та втрати трекінгу перебувають у межах цільових нормативів, що гарантує надійність візуалізації в режимі реального часу.

Аналітичний модуль забезпечив точну агреговану обробку телеметричних даних, формування OLAP-куба та кластеризацію AR-сесій, що дозволило обґрунтовано оцінити стійкість роботи системи за різними користувацькими та сценовими параметрами. Перевірка KPI-показників засвідчила відповідність фактичних значень установленим порогам, що підтверджує ефективність внутрішніх алгоритмів аналізу та збалансованість архітектури системи.

Розгортання програмного комплексу в багатокомпонентній інфраструктурі показало коректну взаємодію AR-клієнта, сервера аналітики та сервера бази даних через захищені канали REST/HTTPS та SQL/TCP. Структура інсталяційного пакета забезпечує відмовостійкість, масштабованість і можливість подальшого розширення системи з мінімальними витратами на конфігурацію.

Узагальнення результатів підтверджує, що реалізована система доповненої реальності відповідає функціональним, технічним і продуктивним вимогам, забезпечує стабільність обчислювальних процесів та надійну роботу в навчально-дослідницькому середовищі, а також має потенціал для подальшого удосконалення й інтеграції з розширеними аналітичними сервісами.

ВИСНОВКИ

У магістерській роботі здійснено повний цикл дослідження, проєктування та реалізації настільної системи доповненої реальності навчально-дослідницького призначення, орієнтованої на візуалізацію та інтерактивні перетворення математичних фігур і поверхонь у реальному просторі. У вступі обґрунтовано актуальність використання AR-технологій у технічних науках, сформульовано мету, завдання, об’єкт, предмет і методи дослідження, а також визначено наукову новизну та практичну цінність розробки.

У першому розділі виконано системний аналіз предметної області, узагальнено сучасні AR-платформи та програмно-апаратні рішення, побудовано UML-моделі процесів і акторів, сформовано функціональні, нефункціональні й технічні вимоги, що дозволило чітко визначити цільовий профіль системи та сформувати постановку завдання.

У другому розділі розроблено логічну модель даних, діаграму класів, діаграми кооперацій, компонентів і пакетів, які формують цілісну архітектурну основу системи: виділено ядро математичних обчислень, підсистеми трекінгу, рендерингу, телеметрії, аналітики та зберігання даних, визначено межі відповідальності модулів і інтерфейси їх взаємодії.

У третьому розділі виконано архітектурну імплементацію AR-системи на базі PyQt6, SQLite, NumPy/SciPy, модулів TrackingEngine з підтримкою ArUco та ORB/AKAZE + PnP/EPnP, Renderer, InteractionService, NotificationCenter, ReportEngine, побудовано структуру core/ui/data/services/plugins, реалізовано асинхронну обробку кадрів, збір і зберігання телеметрії, механізми RBAC, авторизацію, журналювання та експорт результатів у різні формати, що забезпечило повний цикл роботи: ініціалізація сцени, трекінг, генерація й рендеринг об’єктів, взаємодія користувача, аналітика та звітність.

У четвертому розділі проведено модульне й інтеграційне тестування системи, оцінено продуктивність, стабільність трекінгу, якість рендерингу,

коректність збору телеметрії та функціонування OLAP-аналітики й KPI-модулів; отримані результати (FPS не менше 30 при 720p, медіана latency до 50 мс, низькі показники track loss і дрейфу пози, відповідність KPI-цільовим порогам) підтвердили відповідність програмного комплексу сформульованим вимогам, а розгортання у багатокомпонентній архітектурі довело його масштабованість і відмовостійкість.

Мета роботи досягнута повністю, усі поставлені завдання (аналіз предметної області, формування вимог, проєктування архітектури та моделі даних, реалізація програмних модулів, тестування, оцінювання ефективності й опис розгортання) виконано; розроблений прототип AR-системи може використовуватися як інструмент для навчально-дослідницької роботи з математичними моделями та має потенціал подальшого розвитку за рахунок підключення зовнішніх датчиків, розширених аналітичних сервісів і інтеграції з освітніми платформами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Azuma, R. T. *A Survey of Augmented Reality*. Presence: Teleoperators and Virtual Environments, 1997, 6(4), 355–385.
2. Billinghurst, M., Clark, A., & Lee, G. *A Survey of Augmented Reality*. Foundations and Trends in Human–Computer Interaction, 2015, 8(2–3), 73–272.
3. Milgram, P., & Kishino, F. *A Taxonomy of Mixed Reality Visual Displays*. IEICE Transactions on Information and Systems, 1994, E77-D(12), 1321–1329.
4. Schmalstieg, D., & Hollerer, T. *Augmented Reality: Principles and Practice*. Addison-Wesley Professional, 2016.
5. Zhou, F., Duh, H. B. L., & Billinghurst, M. *Trends in Augmented Reality Tracking, Interaction and Display: A Review of Ten Years of ISMAR*. IEEE/ACM International Symposium on Mixed and Augmented Reality, 2008, pp. 193–202.
6. Carmigniani, J., Furht, B., Anisetti, M., Ceravolo, P., Damiani, E., & Ivkovic, M. *Augmented Reality Technologies, Systems and Applications*. Multimedia Tools and Applications, 2011, 51(1), 341–377.
7. Siltanen, S. *Theory and Applications of Marker-Based Augmented Reality*. VTT Technical Research Centre of Finland, 2012.
8. Cawood, S., & Fiala, M. *Augmented Reality: A Practical Guide*. Pragmatic Bookshelf, 2008.
9. Apple Inc. *ARKit 6 Overview*. Retrieved from: https://developer.apple.com/augmented-reality/
10. Google LLC. *ARCore Documentation*. Retrieved from: https://developers.google.com/ar
11. Microsoft Corporation. *HoloLens 2 Technical Overview*. Retrieved from: https://learn.microsoft.com/en-us/hololens
12. PTC Inc. *Vuforia Engine Developer Portal*. Retrieved from: https://developer.vuforia.com/

13. Unity Technologies. *AR Foundation Manual*. Retrieved from: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation

14. Pressman, R. S., & Maxim, B. R. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 2019.

15. Sommerville, I. *Software Engineering*. 10th ed. Pearson Education, 2020.

ДОДАТОК А

Фрагмент програмного коду настільної AR-системи доповненої реальності для візуалізації математичних моделей у технічних науках

```
import sys
import time
import math
import random
import sqlite3
from dataclasses import dataclass
from typing import List, Optional

import numpy as np

from PyQt6.QtCore import (
    Qt,
    QTimer,
    QAbstractTableModel,
    QModelIndex,
)
from PyQt6.QtWidgets import (
    QApplication,
    QMainWindow,
    QWidget,
    QVBoxLayout,
    QHBoxLayout,
    QLabel,
    QPushButton,
```

```
    QTableView,
    QSplitter,
    QFormLayout,
    QLineEdit,
    QComboBox,
    QSpinBox,
    QDockWidget,
    QFrame,
    QMessageBox,
    QStatusBar,
    QCheckBox,
    QFileDialog,
)


# ============================
# 1. Модель даних телеметрії
# ============================

@dataclass
class TelemetryRecord:
    session_id: int
    timestamp: float
    fps: float
    latency_ms: float
    track_loss_pct: float
    drift_deg: float
    severity: str
    message: str
```

```
# ================================
# 2. Менеджер бази даних (SQLite)
# ================================

class DatabaseManager:
    """
    Клас для управління локальною базою даних SQLite.
    Відповідає за створення таблиць, збереження сесій та телеметрії.
    """

    def __init__(self, db_path: str = "ar_telemetry.db") -> None:
        self.db_path = db_path
        self.conn = sqlite3.connect(self.db_path)
        self.conn.row_factory = sqlite3.Row
        self._init_schema()

    def _init_schema(self) -> None:
        cur = self.conn.cursor()

        cur.execute(
            """
            CREATE TABLE IF NOT EXISTS sessions (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                started_at REAL NOT NULL,
                finished_at REAL
            );
            """
        )
```

```
    cur.execute(
        """
        CREATE TABLE IF NOT EXISTS telemetry (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            session_id INTEGER NOT NULL,
            ts REAL NOT NULL,
            fps REAL NOT NULL,
            latency_ms REAL NOT NULL,
            track_loss_pct REAL NOT NULL,
            drift_deg REAL NOT NULL,
            severity TEXT NOT NULL,
            message TEXT NOT NULL,
            FOREIGN KEY (session_id) REFERENCES sessions(id)
        );
        """
    )

    self.conn.commit()

def create_session(self, name: str) -> int:
    cur = self.conn.cursor()
    cur.execute(
        "INSERT INTO sessions (name, started_at) VALUES (?, ?);",
        (name, time.time()),
    )
    self.conn.commit()
    return cur.lastrowid

def close_session(self, session_id: int) -> None:
```

```
    cur = self.conn.cursor()
    cur.execute(
        "UPDATE sessions SET finished_at = ? WHERE id = ?;",
        (time.time(), session_id),
    )
    self.conn.commit()

def save_telemetry(self, record: TelemetryRecord) -> None:
    cur = self.conn.cursor()
    cur.execute(
        """
        INSERT INTO telemetry (
            session_id, ts, fps, latency_ms,
            track_loss_pct, drift_deg, severity, message
        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?);
        """,
        (
            record.session_id,
            record.timestamp,
            record.fps,
            record.latency_ms,
            record.track_loss_pct,
            record.drift_deg,
            record.severity,
            record.message,
        ),
    )
    self.conn.commit()
```

```
        def fetch_telemetry_for_session(self, session_id: int) ->
List[TelemetryRecord]:
            cur = self.conn.cursor()
            cur.execute(
                """
                SELECT session_id, ts, fps, latency_ms,
                       track_loss_pct, drift_deg, severity, message
                FROM telemetry
                WHERE session_id = ?
                ORDER BY ts ASC;
                """,
                (session_id,),
            )
            rows = cur.fetchall()
            return [
                TelemetryRecord(
                    session_id=row["session_id"],
                    timestamp=row["ts"],
                    fps=row["fps"],
                    latency_ms=row["latency_ms"],
                    track_loss_pct=row["track_loss_pct"],
                    drift_deg=row["drift_deg"],
                    severity=row["severity"],
                    message=row["message"],
                )
                for row in rows
            ]


    # ====================================
```

```
# 3. MathCore – генерація математичних
#    поверхонь (спрощена реалізація)
# ======================================


class MathCore:
    """
    Математичне ядро: генерація параметричних поверхонь та кривих.
    У реальній системі дані передаються до модуля рендерингу.
    """

    @staticmethod
    def generate_torus(R: float = 1.0, r: float = 0.4, nu: int = 64, nv: int = 32):
        """
        Генерація точок торуса:
        R – великий радіус, r – малий радіус.
        Повертає масив vertex-координат (x, y, z).
        """
        u = np.linspace(0, 2 * np.pi, nu)
        v = np.linspace(0, 2 * np.pi, nv)
        u, v = np.meshgrid(u, v)

        x = (R + r * np.cos(v)) * np.cos(u)
        y = (R + r * np.cos(v)) * np.sin(u)
        z = r * np.sin(v)

        vertices = np.stack([x, y, z], axis=-1)
        return vertices

    @staticmethod
    def generate_parametric_surface(expr_x: str, expr_y: str, expr_z: str,
```

```
                    umin: float, umax: float,
                    vmin: float, vmax: float,
                    nu: int = 64, nv: int = 64):
    """
    Генерація довільної параметричної поверхні з виразів expr_x, expr_y, expr_z.
    Вирази задаються у термінах змінних u, v.
    """
    u = np.linspace(umin, umax, nu)
    v = np.linspace(vmin, vmax, nv)
    u, v = np.meshgrid(u, v)

    # Обмежене оточення для безпечної інтерпретації виразів
    env = {
        "u": u,
        "v": v,
        "sin": np.sin,
        "cos": np.cos,
        "tan": np.tan,
        "pi": np.pi,
        "sqrt": np.sqrt,
    }

    x = eval(expr_x, {"__builtins__": {}}, env)
    y = eval(expr_y, {"__builtins__": {}}, env)
    z = eval(expr_z, {"__builtins__": {}}, env)

    vertices = np.stack([x, y, z], axis=-1)
    return vertices
```

```
# ================================================
# 4. TrackingEngine – спрощений симулятор трекінгу
# ================================================

class TrackingEngine:
    """
    Підсистема трекінгу. У цьому прототипі – спрощений симулятор,
    який генерує псевдовипадкові значення FPS, latency, track-loss, дрейфу.
    """

    def __init__(self) -> None:
        self._t0 = time.time()

    def sample_metrics(self) -> dict:
        t = time.time() - self._t0

        # Імітація зміни FPS та затримки
        fps = 30.0 + 5.0 * math.sin(t / 5.0)
        latency = 40.0 + 10.0 * math.sin(t / 7.0 + 0.5)

        # Легка флуктуація втрати треку
        track_loss_pct = max(0.0, 1.0 + 1.0 * math.sin(t / 11.0 + 1.0))

        # Дрейф пози
        drift_deg = max(0.2, 1.0 + 0.5 * math.sin(t / 9.0 + 2.0))

        severity = "info"
        message = "Нормальний режим роботи"
```

```
        if fps < 25.0 or latency > 70.0:
            severity = "warning"
            message = "Зниження FPS або підвищена затримка"

        if track_loss_pct > 5.0:
            severity = "error"
            message = "Перевищено поріг втрати трекінгу"

        return {
            "fps": fps,
            "latency_ms": latency,
            "track_loss_pct": track_loss_pct,
            "drift_deg": drift_deg,
            "severity": severity,
            "message": message,
        }


# ============================================================
# 5. Модель для QTableView (телеметрія в інтерфейсі)
# ============================================================

class TelemetryTableModel(QAbstractTableModel):
    """
    Таблична модель для відображення телеметрії у QTableView.
    """

    HEADERS = [
        "Час, с",
        "FPS",
```

```
            "Latency, мс",
            "Track loss, %",
            "Drift, °",
            "Рівень",
            "Повідомлення",
        ]

        def __init__(self, records: Optional[List[TelemetryRecord]] = None) ->
None:
            super().__init__()
            self._records: List[TelemetryRecord] = records or []

        def rowCount(self, parent: QModelIndex = QModelIndex()) -> int:
            return len(self._records)

        def columnCount(self, parent: QModelIndex = QModelIndex()) -> int:
            return len(self.HEADERS)

        def data(self, index: QModelIndex, role: int = Qt.ItemDataRole.DisplayRole):
            if not index.isValid() or role != Qt.ItemDataRole.DisplayRole:
                return None

            rec = self._records[index.row()]
            col = index.column()

            if col == 0:
                return f"{rec.timestamp:.1f}"
            if col == 1:
                return f"{rec.fps:.1f}"
            if col == 2:
```

```
                return f"{rec.latency_ms:.1f}"
            if col == 3:
                return f"{rec.track_loss_pct:.2f}"
            if col == 4:
                return f"{rec.drift_deg:.2f}"
            if col == 5:
                return rec.severity
            if col == 6:
                return rec.message

            return None

        def headerData(self, section: int, orientation: Qt.Orientation,
                       role: int = Qt.ItemDataRole.DisplayRole):
            if role != Qt.ItemDataRole.DisplayRole:
                return None
            if orientation == Qt.Orientation.Horizontal:
                return self.HEADERS[section]
            return section + 1

        def add_record(self, record: TelemetryRecord) -> None:
            self.beginInsertRows(QModelIndex(), len(self._records),
len(self._records))
            self._records.append(record)
            self.endInsertRows()


    # ========================================
    # 6. Виджет сцени (спрощений AR-перегляд)
    # ========================================
```

```
class SceneView(QWidget):
    """
    Спрощений віджет сцени. У реальній системі тут використовуються
    Qt3D або PyOpenGL. У цьому прототипі – лише текстовий статус.
    """

    def __init__(self, parent: Optional[QWidget] = None) -> None:
        super().__init__(parent)
        self.setObjectName("SceneView")
        self.setMinimumHeight(300)

        layout = QVBoxLayout(self)
        layout.setContentsMargins(16, 16, 16, 16)

        self.label_title = QLabel("AR-сцена: візуалізація математичної моделі")
        self.label_title.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.label_title.setStyleSheet("font-size: 16px; font-weight: 600;")

        self.label_status = QLabel("Стан: очікування запуску сесії")
        self.label_status.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.label_status.setStyleSheet("color: #444444;")

        layout.addWidget(self.label_title)
        layout.addWidget(self.label_status)

        frame = QFrame()
        frame.setFrameShape(QFrame.Shape.StyledPanel)
        frame.setStyleSheet("background-color: #202020; border-radius: 8px;")
        frame_layout = QVBoxLayout(frame)
```

```
        frame_layout.setContentsMargins(8, 8, 8, 8)

        frame_caption = QLabel("Тут відображається торус або інша параметрична поверхня")
        frame_caption.setStyleSheet("color: #DDDDDD; font-size: 12px;")
        frame_caption.setAlignment(Qt.AlignmentFlag.AlignCenter)

        frame_layout.addWidget(frame_caption)
        layout.addWidget(frame)

    def set_running(self, running: bool) -> None:
        if running:
            self.label_status.setText("Стан: активна AR-сесія (тестування)")
            self.label_status.setStyleSheet("color: #1e824c;")
        else:
            self.label_status.setText("Стан: зупинено")
            self.label_status.setStyleSheet("color: #aa0000;")


# ========================================
# 7. Виджет керування тестовою сесією
# ========================================

class SessionControlWidget(QWidget):
    """
    Панель керування AR-сесією: назва, старт, стоп, авто-збереження.
    """

    def __init__(self, parent: Optional[QWidget] = None) -> None:
        super().__init__(parent)
```

```
        layout = QFormLayout(self)
        layout.setContentsMargins(8, 8, 8, 8)
        layout.setSpacing(6)

        self.edit_session_name = QLineEdit()
        self.edit_session_name.setText("AR_test_session")

        self.combo_surface = QComboBox()
        self.combo_surface.addItems(
            [
                "Тор ус (R=1.0, r=0.4)",
                "Параметрична поверхня (демо)",
            ]
        )

        self.spin_interval = QSpinBox()
        self.spin_interval.setRange(100, 5000)
        self.spin_interval.setSingleStep(100)
        self.spin_interval.setValue(500)

        self.check_autosave = QCheckBox("Автозбереження телеметрії")
        self.check_autosave.setChecked(True)

        layout.addRow("Назва сесії:", self.edit_session_name)
        layout.addRow("Модель:", self.combo_surface)
        layout.addRow("Інтервал, мс:", self.spin_interval)
        layout.addRow("", self.check_autosave)


# ====================================
```

```
# 8. Головне вікно системи
# ========================================

class MainWindow(QMainWindow):
    """
    Головне вікно настільної AR-системи.
    Об’єднує SceneView, таблицю телеметрії та панель керування.
    """

    def __init__(self) -> None:
        super().__init__()
        self.setWindowTitle("AR-лабораторія для технічних наук")
        self.resize(1200, 700)

        self.db = DatabaseManager()
        self.tracking = TrackingEngine()
        self.current_session_id: Optional[int] = None
        self.t0_session: Optional[float] = None

        central_widget = QWidget()
        central_layout = QVBoxLayout(central_widget)
        central_layout.setContentsMargins(4, 4, 4, 4)
        central_layout.setSpacing(4)

        self.scene_view = SceneView()
        splitter = QSplitter()
        splitter.setOrientation(Qt.Orientation.Vertical)
        splitter.addWidget(self.scene_view)

        self.telemetry_model = TelemetryTableModel()
```

```
        self.telemetry_view = QTableView()
        self.telemetry_view.setModel(self.telemetry_model)
        self.telemetry_view.horizontalHeader().setStretchLastSection(True)
        self.telemetry_view.setAlternatingRowColors(True)

        splitter.addWidget(self.telemetry_view)
        splitter.setStretchFactor(0, 3)
        splitter.setStretchFactor(1, 2)

        central_layout.addWidget(splitter)
        self.setCentralWidget(central_widget)

        # Панель керування (dock)
        self.dock_control = QDockWidget("Керування AR-сесією", self)
        self.dock_control.setAllowedAreas(
            Qt.DockWidgetArea.LeftDockWidgetArea |
Qt.DockWidgetArea.RightDockWidgetArea
        )
        self.session_control = SessionControlWidget()
        dock_container = QWidget()
        dock_layout = QVBoxLayout(dock_container)
        dock_layout.setContentsMargins(8, 8, 8, 8)
        dock_layout.setSpacing(8)

        dock_layout.addWidget(self.session_control)

        self.btn_start = QPushButton("Запустити тестову сесію")
        self.btn_stop = QPushButton("Зупинити сесію")
        self.btn_export = QPushButton("Експорт телеметрії у CSV")
```

```
        btn_row = QHBoxLayout()
        btn_row.addWidget(self.btn_start)
        btn_row.addWidget(self.btn_stop)
        dock_layout.addLayout(btn_row)

        dock_layout.addWidget(self.btn_export)

        dock_container.setLayout(dock_layout)
        self.dock_control.setWidget(dock_container)
        self.addDockWidget(Qt.DockWidgetArea.RightDockWidgetArea,
self.dock_control)

        # Статус-бар
        status = QStatusBar()
        self.setStatusBar(status)

        # Таймер збору телеметрії
        self.timer = QTimer()
        self.timer.timeout.connect(self.collect_telemetry)

        # Сигнали
        self.btn_start.clicked.connect(self.start_session)
        self.btn_stop.clicked.connect(self.stop_session)
        self.btn_export.clicked.connect(self.export_csv)

    # ------------------------
    # Керування сесією
    # ------------------------

    def start_session(self) -> None:
```

```
            if self.current_session_id is not None:
                QMessageBox.warning(
                    self,
                    "Сесія вже активна",
                    "Поточна сесія вже запущена. Спочатку зупиніть її.",
                )
                return

            name = self.session_control.edit_session_name.text().strip()
            if not name:
                QMessageBox.warning(self, "Помилка", "Вкажіть назву сесії.")
                return

            self.current_session_id = self.db.create_session(name)
            self.t0_session = time.time()
            self.telemetry_model = TelemetryTableModel()
            self.telemetry_view.setModel(self.telemetry_model)

            interval = self.session_control.spin_interval.value()
            self.timer.start(interval)

            self.scene_view.set_running(True)
            self.statusBar().showMessage(f"Сесія #{self.current_session_id} запущена")

        def stop_session(self) -> None:
            if self.current_session_id is None:
                return

            self.timer.stop()
```

```
            self.db.close_session(self.current_session_id)
            self.statusBar().showMessage(f"Сесія #{self.current_session_id} завершена")
            self.scene_view.set_running(False)
            self.current_session_id = None
            self.t0_session = None

        # ------------------------
        # Збір телеметрії
        # ------------------------

        def collect_telemetry(self) -> None:
            if self.current_session_id is None or self.t0_session is None:
                return

            metrics = self.tracking.sample_metrics()
            t_rel = time.time() - self.t0_session

            rec = TelemetryRecord(
                session_id=self.current_session_id,
                timestamp=t_rel,
                fps=metrics["fps"],
                latency_ms=metrics["latency_ms"],
                track_loss_pct=metrics["track_loss_pct"],
                drift_deg=metrics["drift_deg"],
                severity=metrics["severity"],
                message=metrics["message"],
            )

            # Додавання до таблиці
```

```
        self.telemetry_model.add_record(rec)

        # За потреби – збереження в БД
        if self.session_control.check_autosave.isChecked():
            self.db.save_telemetry(rec)

        # Оновлення статусу
        self.statusBar().showMessage(
            f"FPS={rec.fps:.1f}, latency={rec.latency_ms:.1f} мс, "
            f"track_loss={rec.track_loss_pct:.2f} %, drift={rec.drift_deg:.2f}°"
        )

    # -------------------------
    # Експорт у CSV
    # -------------------------

    def export_csv(self) -> None:
        if self.current_session_id is None:
            QMessageBox.warning(
                self,
                "Немає активної сесії",
                "Спочатку запустіть і завершить хоча б одну сесію.",
            )
            return

        records = self.db.fetch_telemetry_for_session(self.current_session_id)

        if not records:
            QMessageBox.information(
                self,
```

```
            "Немає даних",
            "Для поточної сесії немає телеметрії для експорту.",
        )
        return

    path, ok = QFileDialog.getSaveFileName(
        self,
        "Зберегти телеметрію у CSV",
        f"session_{self.current_session_id}.csv",
        "CSV файли (*.csv)",
    )

    if not ok or not path:
        return

    try:
        import csv

        with open(path, "w", newline="", encoding="utf-8") as f:
            writer = csv.writer(f, delimiter=";")
            writer.writerow(
                [
                    "session_id",
                    "timestamp_s",
                    "fps",
                    "latency_ms",
                    "track_loss_pct",
                    "drift_deg",
                    "severity",
                    "message",
```

```
                ]
            )
            for r in records:
                writer.writerow(
                    [
                        r.session_id,
                        f"{r.timestamp:.3f}",
                        f"{r.fps:.3f}",
                        f"{r.latency_ms:.3f}",
                        f"{r.track_loss_pct:.3f}",
                        f"{r.drift_deg:.3f}",
                        r.severity,
                        r.message,
                    ]
                )

            QMessageBox.information(
                self,
                "Експорт завершено",
                f"Телеметрію успішно збережено у файл:\n{path}",
            )
        except Exception as e:
            QMessageBox.critical(self, "Помилка експорту", str(e))


# ====================================
# 9. Точка входу в програму
# ====================================

def main() -> None:
```

```
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())


if __name__ == "__main__":
    main()
```