

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО

В.о. завідувача кафедри
Олександр КОВАЛЬ

«__» _____ 2026 р.

Дипломна робота

**на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення**

**на тему: «Програмний додаток для підбору рекуператора методами
машинного навчання на основі цифрового двійника приміщення»**

Виконав

студент IV курсу, групи ТВ-21

Гранківський Дмитро Петрович

(прізвище, ім'я, по батькові)

(підпис)

Керівник:

Асистент Швайко Валерій Григорович

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент:

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень із праць інших авторів
без відповідних посилань.

Студент

(підпис)

Київ – 2026

**Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ

«___» _____ 2026р.

ЗАВДАННЯ

на дипломну роботу студенту

_____ Гранківському Дмитру Петровичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Програмний додаток для підбору рекуператора методами
машинного навчання на основі цифрового двійника приміщення

керівник роботи Швайко Валерій Григорович, асистент кафедри ІПЗЕ

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «28» травня 2026р. №1992-с

2. Строк подання студентом роботи «10» червня 2026р.

3. Вихідні дані до роботи: мова програмування Python, середовище розробки
PyCharm

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно
розробити): проаналізувати існуючі системи; розробити цифровий двійник
приміщення; навчити ШІ-агента; використати його для підбору рекуператора

5. Перелік ілюстративного матеріалу: приклади використання програмного
застосунку, приклади роботи аналогів, діаграми програмної застосунку

6. Дата видачі завдання «31» жовтня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	31.10.2025	
2	Дослідження предметної області	01.11.2025 – 31.11.2025	
3	Дослідження існуючих рішень	01.12.2025 – 31.12.2025	
4	Постановка вимог до проєктування системи	01.01.2026 – 31.01.2026	
5	Розробка програмного продукту	01.02.2026 – 03.05.2026	
6	Тестування	04.05.2026 – 10.05.2026	
7	Захист програмного продукту	11.05.2026 – 15.05.2026	
8	Оформлення дипломної роботи	18.05.2026 – 31.05.2026	
9	Передзахист	01.06.2026 – 05.06.2026	
10	Захист	15.06.2026 – 19.06.2026	

Студент

(підпис)Дмитро ГРАНКІВСЬКИЙ

(ім'я, прізвище)

Керівник роботи

(підпис)Валерій ШВАЙКО

(ім'я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 69 сторінок, 19 рисунків, 2 додатки та 16 посилань.

Актуальність теми. Підвищення герметизації будівель задля енергоефективності погіршує якість повітря через накопичення забруднювачів. Статичні розрахунки не враховують динаміку експлуатації приміщень, що зумовлює потребу в інтелектуальних системах для точного підбору вентиляції.

Метою роботи є розробка програмного додатка для автоматизованого підбору моделей рекуператорів на основі цифрового двійника приміщення з використанням методів машинного навчання.

Об'єктом дослідження є процес автоматизованого підбору та керування режимами роботи припливно-витяжних вентиляційних систем із рекуперацією тепла в приміщеннях.

Предметом дослідження є архітектури, моделі, алгоритми навчання з підкріпленням та сурогатного моделювання для аналізу мікроклімату та вибору вентиляційного обладнання.

Методи дослідження включають технологію симуляції EnergyPlus, об'єктно-орієнтоване проєктування (патерни Фасад, Адаптер), алгоритми LightGBM для сурогатного моделювання та навчання з підкріпленням PPO у середовищі Gymnasium.

Практичне значення одержаних результатів полягає у створенні веб-застосунку на базі Streamlit з модулем валідації. Система дозволяє без спеціальних знань конструювати цифровий двійник приміщення, миттєво розраховувати динаміку мікроклімату за допомогою сурогатного рушія та знаходити оптимальні рішення для вентиляції.

Ключові слова: цифровий двійник, рекуператор тепла, мікроклімат, навчання з підкріпленням, алгоритм PPO, сурогатна модель, EnergyPlus, мова програмування Python.

ABSTRACT

Structure and scope of the thesis. The work contains 69 pages, 19 figures, 2 appendices, and 16 references.

Relevance of the topic. Increasing the sealing of buildings for energy efficiency worsens air quality due to the accumulation of pollutants. Static calculations do not take into account the dynamics of the operation of premises, which determines the need for intelligent systems for precise selection of ventilation.

The purpose of the work is the development of a software application for automated selection of heat recovery models based on a digital twin of the room using machine learning methods.

The object of research is a process of automated selection and control of operating modes of supply and exhaust ventilation systems with heat recovery in rooms.

The subject of the work are architectures, models, reinforcement learning algorithms and surrogate modeling for microclimate analysis and ventilation equipment selection.

Research methods include EnergyPlus simulation technology, object-oriented design (Facade, Adapter patterns), LightGBM algorithms for surrogate modeling, and PPO reinforcement learning in the Gymnasium environment.

The practical value consists of creating a web application based on Streamlit with a validation module. The system allows you to construct a digital twin of a room without special knowledge, instantly calculate the dynamics of the microclimate using a surrogate engine, and find optimal ventilation solutions.

Keywords: digital twin, heat recovery unit, microclimate, reinforcement learning, PPO algorithm, surrogate model, EnergyPlus, Python programming language.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	8
ВСТУП.....	9
1. АНАЛІЗ ТЕХНОЛОГІЙ МОДЕЛЮВАННЯ МІКРОКЛІМАТУ ТА МЕТОДІВ ОПТИМІЗАЦІЇ.....	10
1.1. Проблема забезпечення мікроклімату та роль рекуперації в інженерних лабораторіях.....	10
1.2. Технологія цифрових двійників та аналіз інструментів термодинамічної симуляції будівель.....	12
1.3. Особливості застосування навчання з підкріпленням у задачах автоматизації інженерних систем.....	14
Висновки до розділу 1	16
2. ПРОЄКТУВАННЯ СТРУКТУРИ ТА ОБҐРУНТУВАННЯ КОМПОНЕНТІВ СИСТЕМИ HIVANALYZER.....	18
2.1. Функціональні вимоги та загальна архітектура програмного комплексу	18
2.2. Формалізація об'єктів цифрового двійника приміщення та моделювання внутрішнього навантаження	19
2.3. Проєктування динаміки забруднювачів та системи вентиляції	21
2.4. Архітектура програмного інтерфейсу взаємодії з симулятором.....	23
Висновки до розділу 2	24
3. Програмна реалізація комплексу HIVANALYZER.....	26
3.1. Організація кодової бази та структура каталогу проєкту	26
3.2. Програмна реалізація пакета координування симуляцій.....	32
3.3. Реалізація обчислювального ML-контурі програми.....	36

3.4. Програмна реалізація логіки вибору моделей та алгоритму PPO.....	43
3.5. Програмний код інтерактивного Streamlit UI	44
Висновки до розділу 3	47
4. ВЕРИФІКАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ ТА ІНСТРУКЦІЯ КОРИСТУВАЧА	48
4.1. Автоматизоване модульне тестування та аналіз складності й підтримованості вихідного коду.....	48
4.2. Сценарій роботи дослідника: підготовка синтетичних даних та навчання моделей машинного навчання	52
4.3. Сценарій роботи користувача: архітектурне конструювання приміщення та експорт конфігурацій	56
4.4. Проведення автоматизованого підбору обладнання та аналіз радарних діаграм реалістичності мікроклімату	60
Висновки до розділу 4	65
ВИСНОВКИ.....	66
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI	Artificial Intelligence - штучний інтелект
API	Application Programming Interface - інтерфейс програмування застосунків
CPU	Central Processing Unit - центральний процесор
CSV	Comma-Separated Values - текстовий табличний формат даних із розділенням комами
EMS	Energy Management System - система енергетичного менеджменту (вбудований алгоритмічний рушій EnergyPlus)
ETL	Extract, Transform, Load - процес витягування, трансформації та завантаження даних
GPU	Graphics Processing Unit - графічний процесор
HVAC	Heating, Ventilation, and Air Conditioning - опалення, вентиляція та кондиціонування повітря
IDF	Input Data File - конфігураційний файл вхідних даних EnergyPlus
JSON	JavaScript Object Notation - текстовий формат обміну даними
LightGBM	Light Gradient Boosting Machine - легка машина градієнтного бустингу дерев рішень
MAE	Mean Absolute Error - середня абсолютна похибка
ML	Machine Learning - машинне навчання
PPO	Proximal Policy Optimization - алгоритм проксимальної оптимізації політики навчання з підкріпленням
RL	Reinforcement Learning - навчання з підкріпленням
RMSE	Root Mean Squared Error - середньоквадратична похибка
UI	User Interface - інтерфейс користувача
XAI	Explainable Artificial Intelligence - пояснювальний штучний інтелект

ВСТУП

Сучасний етап розвитку енергетичного сектору в Україні, зумовлений дефіцитом енергоносіїв, почав тяжіти до високої енергоефективності будівель та споруд. Для виробничих та наукових лабораторій це питання стає критичним, оскільки окрім забезпечення енергоефективності, обладнання має підтримувати суворі вимоги до якості повітря в приміщенні та температурних режимів через вплив на безпеку праці та результати експериментів. Класичні системи вентиляції працюють за реактивним принципом або за строго заданим графіком і в обох випадках мають великий вплив на утримання тепла в середині приміщення, що в результаті призводить до збільшення витрат енергії на системи мікроклімату.

Ефективним та популярним рішенням цієї проблеми є встановлення припливно-витяжних систем вентиляції з рекуперацією тепла. Підбір такого обладнання може стати викликом, адже ставить перед інженером складну задачу з багатьма змінними такими як забруднення повітря лабораторним приладдям, а також антропогенний та кліматичний вплив на температурні показники приміщення. Традиційні програмні комплекси хоч і забезпечують високу точність обчислень, проте вимагають глибокого розуміння тепло-енергетичних процесів та навичок користування таким програмним забезпеченням. Саме на вирішення цієї проблеми і спрямоване дане дослідження.

Поєднання технології цифрових двійників та машинного навчання дозволить створити систему, що може симулювати простір, в якому працюватиме обладнання, відтворюючи різні сценарії, а також підставити в ці умови різні моделі рекуператорів. Таким чином можна виявити критичні обставини та побачити в які моменти те чи інше вентиляційне обладнання не зможе виконувати свою функцію належним чином. В той же час програмне забезпечення оптимізує цей процес та дозволить інженеру отримати якісний результат лише задавши параметри приміщення та натиснувши декілька кнопок.

1 АНАЛІЗ ТЕХНОЛОГІЙ МОДЕЛЮВАННЯ МІКРОКЛІМАТУ ТА МЕТОДІВ ОПТИМІЗАЦІЇ

Перед початком створення програмного забезпечення необхідно провести аналітичний огляд предметної області, визначити ключові інженерні протиріччя та сформулювати теоретичний базис, навколо якого можна моделювати кіберфізичну систему. Для даного дослідження необхідно розуміти фізику приладів контролю мікроклімату, особливості використання цифрового двійника приміщення, а також визначити найефективніші алгоритми та методи аналізу та прогнозування даних. Особливу увагу необхідно приділити оптимізації машинного навчання та взаємодії ІІІ-агентів з цифровими на фізичними середовищами.

1.1 Проблема забезпечення мікроклімату та роль рекуперації в інженерних лабораторіях

Сучасні тенденції в архітектурі та будівництві тяжіють до максимальної герметизації приміщень задля збільшення енергоефективності. Проте порушення природного повітрообміну має значний негативний вплив на якість внутрішнього повітря. Основним індикатором антропогенного забруднення є рівень вуглекислого газу в повітрі. Дослідження показують, що рівень CO₂ більше 1000ppm [1] спричиняє зниження когнітивних здібностей, впливає на почуття підвищене втоми та знижує імунітет. А для спеціалізованих приміщень, як то інженерні лабораторії чи виробничі цехи, додається фактор ризику забруднення дрібнодисперсним пилом та органічними сполуками, що виділяються під час таких процесів як 3D-друк та пайка.

Традиційним рішенням такої проблеми є звичайне провітрювання, але такий метод нівелює будь-які вигоди від герметизації приміщення, адже випускає акумульоване тепло з середини приміщення назовні. Оскільки на системи забезпечення мікроклімату припадає від 40% до 60% загальних витрат енергії, це

створює компроміс: для швидкого очищення повітря потрібно збільшити кратність повітрообміну, що призводить до значного викиду тепла з приміщення, тим жертвуючи енергоефективністю. Найкращим методом вирішення цього протиріччя є система припливно-витяжної вентиляції з рекуперацією тепла. Рекуператор тепла – це пристрій який передає тепло від витяжного повітря припливному без їхнього змішування. Таким чином через поверхню теплообмінника вдається зберегти близько 80 – 90% тепла приміщення. Це забезпечує підтримку якості внутрішнього повітря не порушуючи тим самим термодинамічний баланс будівлі.

Фізичний принцип рекуперації базується на законах термодинаміки та теплообміну [2]. Повітря рухається через систему суміжних каналів, розділених тонкими пластинами матеріалу з високою теплопровідністю, зазвичай мідь або алюміній. Залежно від напрямків руху потоків та конструкції каналів рекуператори поділяються на перехресноточні, протиточні та ротаційні. Для інженерної лабораторії доцільніше використовувати перехресноточні або протиточні пристрої, оскільки вони унеможливають змішування та перетікання повітря між припливними та витяжними потоками.

Разом з усіма плюсами використання рекуператора з'являється також проблематика вибору конкретної моделі, адже це багатofакторна інженерна задача. Задля точного підбору оптимальних параметрів потрібно враховувати не лише об'єм повітря, динаміку експлуатації, кількість забрудників та графік присутності людей в приміщенні. Статичні обрахунки часто не враховують такі передумови, що призводить до встановлення обладнання що не справляється з піковими навантаженнями, або ж до надмірних інвестицій в надто потужне обладнання. Вирішення цієї проблеми і бере за мету дане дослідження.

1.2 Технологія цифрових двійників та аналіз інструментів термодинамічної симуляції будівель

Основою сучасних інтелектуальних систем для керування інженерною інфраструктурою є концепція цифрових двійників [3]. Згідно з теорією кіберфізичних систем, цифровий двійник – це трикомпонентна структура, що включає в себе реальний об'єкт у фізичному просторі, його віртуальну копію відтворену в цифровому просторі та канали зв'язку між ними. На відміну від традиційної симуляції такий двійник є динамічною віртуальною реплікацією фізичного об'єкта чи процесу, що здатна в реальному часі або в процесі симуляції відтворювати поведінку чи реагувати на зміни зовнішніх чинників, враховуючи такі параметри як теплова інерція стін, пристроїв, а також мікровитоки повітря через тріщини чи отвори в поверхнях огорожувальних конструкцій. В контексті цього дослідження, використання цифрового двійника виступає не лише засобом зручної візуалізації, а й проєктування та верифікації алгоритмів. Така технологія допомагає створити систему керування та симуляції нового покоління, які оптимізують роботу обладнання за рахунок прогнозування поведінки реального об'єкта.

Для розробників програмного забезпечення створення цифрових двійників також вирішує проблему дефіциту високоточних даних. Для задачі автоматизації та аналізу систем вентиляції чи обігріву приміщень збір реальних даних забирає тривалий період часу та потребує значних фінансових інвестицій у сенсорну мережу. Генерація синтетичних даних за допомогою цифрового двійника дозволяє нівелювати ці проблеми створенням контрольованого точного середовища для збору якісного датасету за короткий період часу. Це особливо важливо для розробки систем машинного навчання, адже об'єм навчальної вибірки напряму впливає на точність прогнозів моделі.

Особливе місце концепція цифрових двійників посідає в системах навчання з підкріпленням [4]. Такі моделі вимагають наявності зворотного зв'язку між

моделлю та даними, адже агенту в процесі навчання потрібно миттєво бачити результати своїх змін. Таким чином цифрові двійники стають ключовим етапом ланцюжка розробки енергетичного програмного забезпечення, дозволяючи перехід від статичного обчислення до реакційних аналітичних систем.

Вибір інструментарію для створення цифрового двійника є важливим кроком в розробці, оскільки він визначає точність синтетичних даних та можливість автоматизації певних процесів інтеграцією його як рушія в програмне забезпечення. Серед готових рішень для енергетичного моделювання було обрано EnergyPlus [5], як найбільш придатне середовище для такого роду симуляцій. Це відкрите кросплатформне програмне забезпечення, розроблене за підтримки міністерства енергетики США, яке вже встигло стати галузевим стандартом для моделювання теплових та повітряних процесів у приміщенні.

Ключовою властивістю EnergyPlus для цього проєкту є використання інтегрованого методу балансу тепла до маси для розрахунку стану повітря в конкретних зонах. Це дозволяє програмному забезпеченню враховувати взаємодію огорожувальних конструкцій, систем мікроклімату, внутрішніми тепловиділеннями та рештою факторів впливу на температуру та якість повітря, що є значною перевагою над багатьма системами, що працюють за спрощеними принципами.

Математичне ядро EnergyPlus базується на одночасному розв'язанні рівнянь теплового балансу внутрішніх та зовнішніх поверхонь приміщення разом з рівняннями балансу повітряної маси приміщення. Це якісно відрізняє його від застарілих методів зважуваних коефіцієнтів, оскільки такий метод балансу тепла не покладається на лінійні зміни в процесах, що дозволяє чітко та точно обраховувати радіаційний та конвекційний теплообмін приміщення з навколишнім середовищем. Це особливо важливо для лабораторії SmartEnergy, оскільки вона має нетипову Г-подібну форму та відмежована стінами з різних матеріалів.

З архітектурної точки зору EnergyPlus має наступні переваги:

1) Текстовий формат вхідних даних. Для запуску симуляцій використовується текстовий маніфест усіх параметрів та сценаріїв у специфічному форматі IDF. Це дозволяє легко інтегрувати його в стороннє програмне забезпечення створивши IDF-міст – модуль, що генеруватиме та модифікуватиме симуляції.

2) Консольний інтерфейс. EnergyPlus може функціонувати як обчислювальне ядро, без графічного інтерфейсу. Це дозволяє перетворити його зі стороннього ПЗ на русій симуляцій та безшовно інтегрувати у власне програмне забезпечення.

3) Висока часова дискретність. Можливість зменшити часовий проміжок симуляції до 1 хвилини дозволяє значно підвищити точність симуляції, детально проаналізувати критичні стани системи мікроклімату та відстежити перехідні стани, такі як різке забруднення повітря під час технічних операцій.

Таким чином використання EnergyPlus в поєднанні з програмними бібліотеками дозволяє повністю інтегрувати його як складову програмного забезпечення, що розробляється, та забезпечує гнучкість в керуванні симуляціями, перетворивши модуль обрахунку на інтерактивне середовище та джерело високонадійних даних, які в подальшому стануть критичними та необхідними для якісного навчання моделей машинного навчання для аналізу та прогнозування фізичних процесів усередині приміщення.

1.3 Особливості застосування навчання з підкріпленням у задачах автоматизації інженерних систем

Традиційні підходи до керування інженерними системами будівництва зазвичай базуються на чітких жорстко заданих правилах та алгоритмах. Це перевірені робочі методи, однак в умовах високої нелінійності, яка притаманна сучасним енергоефективним приміщенням, точність та ефективність таких

підходів значно падає. Впровадження методів машинного навчання дозволяє перейти до предикатного керування, в якому система здатна адаптуватися та реагувати на зміни середовища. У контексті керування системами рекуперації варто виділити два основні класи методів машинного навчання:

Навчання з учителем. Оскільки запуск повноцінної симуляції вимагає достатньо багато обчислювального ресурсу, використання алгоритмів регресії дозволяє передбачити приблизний результат симуляції. У межах даного дослідження навчання з учителем використовується для створення так званої сурогатної моделі. Оскільки запуск обчислювального ядра EnergyPlus займає 1-2 секунди, після чого отримані дані потрібно прочитати з текстового файлу та нормалізувати, навчання інших моделей використовуючи лише його як джерело даних стає практично неможливим, оскільки більшості алгоритмів потрібно десятки тисяч кроків для нормального навчання. Використання ансамблевого навчання, такого як Random forest, дозволяє створити модель апроксимації, що може виконувати ті ж задачі, що покладаються на обчислювальне ядро в тисячі разів швидше, але лише приміщення з аналогічними параметрами до того, на симуляції якого модель була навчена. Така економія ресурсів дозволяє прискорити отримання результатів, що в деяких сценаріях є критичною метрикою алгоритму.

Навчання з підкріпленням. Це є найперспективнішим методом для задач автоматизації вибору, особливо за наявності цифрового двійника. Алгоритм має можливість ітераційної роботи з симуляцією, що дає йому доступ до внутрішніх випробувань та експериментів всередині навчального процесу. Такий підхід до навчання дає можливість в межах короткого періоду часу чи не надто варіативного датасету розглянути величезну кількість можливостей та комбінацій, що значно підвищує точність результатів. Математичним фундаментом навчання з підкріпленням є марковські процеси прийняття рішень. Так задача управління мікрокліматом подається як кортеж, що складається з простору станів, дій, а також ймовірностей переходів та функції винагороди. Модель не отримує правильних дій та результатів, а натомість має контроль над взаємодією з середовищем та простір

до експериментів. Після прийняття рішення алгоритм отримує результати своїх дій, їх оцінку в форматі штрафу чи винагороди, що дозволяє агенту вчитися на власних помилках для прийняття правильних рішень задля вирішення поставленої задачі.

Особливу увагу зараз приділяють алгоритмам сімейства Policy Gradient, зокрема алгоритму Proximal policy optimization [6]. Цей алгоритм вирізняється серед решти стабільністю в навчанні, а також можливістю працювати у багатовимірному просторі, себто мати багато окремих входних параметрів. Для термо- та аеродинамічних задач це є надзвичайно важливою якістю. Ця теоретична перевага досягається використанням цільової функції з механізмами обмеження. У класичних методах побудови градієнта стратегій крок змін може виявитися занадто великим, що в результаті призводить до збою політики поведінки агента, після чого йому потрібно майже наново визначати вагові коефіцієнти. PPO створює безпековий коридор значень тим самим обмежуючи амплітуду змін. Такий підхід забезпечує монотонне прогнозоване навчання, яке місцями повільніше за інші методи, але у випадку нелінійних складних систем, як то прогнозування термодинамічних змін, дозволяє уникнути надто різких стрибків. Тим самим алгоритм зменшує рівень хаосу під час навчання та уникає падінь в аварійні ситуації системи.

Таким чином інтеграція машинного навчання в проєкт перетворює програмне забезпечення з пасивного інструменту, на активну систему прийняття рішень. Аналіз показує, що поєднання високоточного обрахунку EnergyPlus разом з гнучкістю алгоритмів машинного навчання є найбільш раціональним підходом до вирішення проблеми дослідження.

Висновки до розділу 1

В першому розділі проведено аналітичне дослідження технологій контролю мікроклімату приміщення, тренди та зміни в цій сфері. Також розглянуто методи та практики підбору вентиляційного обладнання з рекуперацією тепла. Було

визначене математичне ядро проєкту у вигляді програми EnergyPlus та вивчено його структуру та підходи обрахунків. Проведено порівняльний аналіз різних підходів до побудови навчальних середовищ та алгоритмів машинного навчання. Визначені оптимальні методи створення та навчання ШІ-агентів для якісного виконання поставлених задач та отримання точних результатів.

2 ПРОЄКТУВАННЯ СТРУКТУРИ ТА ОБҐРУНТУВАННЯ КОМПОНЕНТІВ СИСТЕМИ HIVANALYZER

2.1 Функціональні вимоги та загальна архітектура програмного комплексу

Програмний комплекс проєктується як інтелектуальна система для спрощення та оптимізації процесу вибору коректного вентиляційного обладнання. Відповідно до такого програмного забезпечення є перелік функціональних та архітектурних вимог, як от: забезпечення зручного графічного введення даних приміщення для цифрового двійника, експорт та збереження налаштувань і моделей машинного навчання, автоматизована генерація файлів конфігурації математичного симулятора, фоновий запуск обчислювального ядра та забезпечення безпомилкової обробки отриманих результатів, формування збалансованих, насичених датасетів для навчання ШІ-агентів, оптимізація навчання до заліза для реалістичних термінів навчання моделей.

Для реалізації цих та подібних вимог було прийнято рішення створення модульної архітектури додатка, що базується на принципах розмежування відповідальності. Найвищим рівнем застосунку має стояти реактивний інтерфейс, який забезпечується фреймворком Streamlit [7] через його гнучкість та універсальність. Оскільки в цьому фреймворку фронт-енд додатку вимальовується мовою Python, це дозволило організувати зручну передачу та обробку даних між модулями додатку та застосувати широкі та версатильні бібліотеки для максимально зручної та зрозумілої візуалізації масивів даних в графіках та інтерактивних таблицях.

Нижній рівень застосунку складається з ізольованих функціональних модулів. Ці модулі розбиті на два умовних пакети. Перший – пакет симуляції. Він відповідає за взаємодію з обчислювальним ядром, обробку вхідних даних з інтерфейсу, складання маніфесту симуляції для EnergyPlus, обробку та аналіз

отриманих даних, формування різних сценаріїв поведінки фізичного середовища генерації якісного датасету. Другий – пакет штучного інтелекту. Його задача полягає в створенні навчального середовища для ШІ-агентів, задання математичних абстракцій для апроксимації обчислень для сурогатної моделі на базі Random Forest, конфігурації Gymnasium [8] середовища та нейромережевої структури алгоритму PPO. Таке архітектурне розподілення дозволило зробити додаток витривалим до збоїв, забезпечити оптимальне розмежування даних між модулями та відокремити логіку розділів для зручної навігації розробника під час розробки та тестування.

2.2 Формалізація об'єктів цифрового двійника приміщення та моделювання внутрішнього навантаження

Початковим етапом побудови цифрового двійника як для об'єктів енергетики, так і для цивільних споруд є визначання математичної та алгоритмічної формалізації простору. У межах розробки програмного комплексу HIValyzer постала задача перенесення реальних геометричних параметрів кімнати у структуровані дані, які будуть правильно інтерпретуватися обчислювальним ядром. Для цього необхідно відійти від принципу фізичних об'єктів як то стін чи підлоги на користь теплових зон, кожна з яких описується набором поверхонь з чіткими параметрами координат у фізичному просторі та фізичними властивостями.

Першим кроком стало визначення геометричних параметрів приміщення та створення точної моделі, що відтворює реальний об'єкт дослідження. Лабораторія SmartEnergyLab має складну Г-подібну будову, що ускладнює роботу, адже часто такі кімнати апроксимуються до прямокутника з аналогічною площею та об'ємом. Проте EnergyPlus може працювати зі складними формами, тому важливо правильно розділити теплові зони для коректної роботи симулятора. Лабораторія має загальну площу 60.8 м² з однією зовнішньою стіною довжиною 11.75 м. Висота стель в

кімнаті 3.25 м, звідси об'єм лабораторії дорівнює 197.6 м³. На зовнішній стіні розташовані два двостулкові вікна, загальною площею скляної поверхні 21.73 м². Кілька внутрішніх стін побудовані з дерева та скла, що також потрібно враховувати під час створення моделі, адже такі стіни мають меншу герметичність та слабо акумулюють тепло.

Розраховуючи рух повітряної маси також важливо звернути увагу на Г-подібну форму приміщення. Така будова додає кілька зовнішніх та внутрішніх кутів, що впливає на вектори руху повітря, що може створювати ефекти масового та теплового застою. Для спрощення прогнозування таких явищ стіни було спроектовано як суцільних плоский полігон у тривимірній системі координат.

Програмна специфікація поверхонь задається як низькорівневий об'єкт, що генерується через кути поверхні, кожен з яких має свою трійку координат x , y та z відносно обраного локального нуля кімнати. Важливою вимогою обчислювального ядра є дотримання обходу вершин поверхні проти годинникової стрілки для коректного визначення вектору нормалі стіни. Це необхідно для розрахунку кутів падіння сонячного проміння та теплопередачі стін шляхом випромінювання.

Такий детальний опис огорожувальних конструкцій дозволив зібрати точну копію приміщення та передати її в IDF маніфест для подальшої симуляції. Великою перевагою EnergyPlus над спрощеними моделями є можливість розглядати кожну стіну як комплексну структуру, що може складатися з різних матеріалів та мати унікальні геометричні та фізичні властивості. Такий підхід дозволив іще на початкових етапах уникнути проблеми недостатньої теплової інерції. Для перетворення статичної моделі в динамічну було розроблено систему розкладів та сценаріїв за допомогою об'єкта Schedule:Compact, що дозволило зімітувати реальні процеси, що відбуваються всередині лабораторії, а не просто симулювати порожню коробку. Основними чинниками забруднення було обрано атропогенний вплив та технічне обладнання. Антропогенне забруднення впливає на рівень CO₂ в приміщенні та на кількість генерованого тепла. Оскільки при легкій офісній роботі одна людина в середньому виділяє 120-150 Вт тепла та близько 40 л вуглекислого

газу на годину, цей показник важливо враховувати. За умови проведення заходу на 10-15 людей всередині лабораторії, що для навчального закладу є цілком реальним сценарієм, кількість тепла, що генеруватимуть люди можна порівняти з невеликим обігрівачем, а рівень вуглекислого газу – з невеликим бензиновим генератором. Це особливо відчутно влітку, коли скидання тепла в навколишнє середовище стає менш ефективним та вимагає застосування додаткового обладнання окрім звичайної вентиляції. Щодо технологічного обладнання, тепловиділення від персональних комп'ютерів, паяльних станцій, мікрокліматичного обладнання тощо також необхідно враховувати в обрахунки. Причому важливо не просто додати їх як джерело тепла чи забруднення, а коректно зазначити їх графіки роботи. Так, наприклад, кондиціонер може працювати безперервно допоки в лабораторії є люди, а от паяльна станція вмикається зазвичай не короткі проміжки часу і не завжди регулярно.

Для зручності організації цих та інших структур було розроблено спеціальне місце зберігання констант у модулі `core/constants.py`, де типізовані та структуровані всі фундаментальні параметри. Будь-який аналітичний чи симуляційний модуль програми імпортує параметри саме з цього файлу. Таким чином забезпечено математичну строгість та постійність обрахунків та симуляцій.

2.3 Проектування динаміки забруднювачів та системи вентиляції

Для створення цифрового двійника інженерної лабораторії важливим аспектом є також побудова моделі повітряного масообміну, що враховує не лише температуру, а й хімічні та фізичні властивості середовища. Реалістичне симулювання вентиляції вимагає чіткого визначення джерел забруднення, їх інтенсивності в конкретні часові періоди та способу поширення приміщенням.

Ключовою особливістю розробленого цифрового двійника є моделювання нелінійних чинників забруднення повітря. Оскільки в симуляторі враховується не

лише рівень вуглекислого газу, а й викиди різного роду забрудників у вигляді дрібнодисперсного пилу, важливо мати правильні інструменти керування таким фактором впливу на середовище. Необхідно враховувати не лише тривалість забруднення, але і його інтенсивність, швидкість осідання та місця концентрації.

Для управління такими факторами використовується об'єкт `ZoneContaminantSourceAndSink:Generic:Continuous`. Він дозволяє регулювати інтенсивність забруднення згідно розкладу. Таким чином отримується не лише стабільне відтворення невеликої дози забруднення, адже лабораторія не є стерильною, а й з'являється можливість задавати конкретні регулярні сценарії, або створювати стрес-тести для аналізу поведінки системи вентиляції в екстремальних ситуаціях.

Для ефективного реагування на нелінійні чинники забруднення в програмному модулі реалізовано дворівневу систему керування даними. Ця система поєднує фізичні параметри обладнання з алгоритмічно заданими сценаріями, що забезпечує передачу в симулятор наближені до реальності дані для правдоподібної симуляції робочих процесів у лабораторії. Ця система розбита на такі модулі: фізичний блок, програмна логіка та інтерфейс передачі даних. Фізичний блок використовує об'єкт `ZoneHVAC:EnergyRecoveryVentilator`, який описує фізичні характеристики рекуператора та дозволяє їх змінювати. Програмна логіка виконує роль контролера роботи рекуператора. За допомогою `Energy Management System` та чітко розписаних сценаріїв, симулятор дозволяє отримувати стабільну та точну реакцію як на зміни умов приміщення, так і на часові показники. Інтерфейс передачі даних – це модуль `simulation_engine.py`, що займається оркестрацією симуляцій. Він передає дані до IDF-моста, після чого запускає симуляцію за згенерованим маніфестом, а потім парсить отримані результати за допомогою бібліотеки `Pandas` [9] у файл `eplusout.csv` для подальшої передачі їх іншим аналітичним модулям проєкту.

2.4 Архітектура програмного інтерфейсу взаємодії з симулятором

Розробка інтерфейсу між високорівневим кодом програми та обчислювальним ядром є важливим етапом розробки, адже забезпечує цілісність та коректність роботи цифрового двійника всередині додатку. Цей процес вимагає побудови надійної програмної абстракції для усунення прямої залежності між процесами високорівневого коду та низькорівневими інженерними обчисленнями. Такий підхід до архітектури проєкту описується як патерн фасаду та адаптера.

В модулі `simulation` створено обидва патерни. Фасад дає доступ решті модулів програми до зрозумілих точок входу та відформатованих даних. Оскільки графічний інтерфейс написаний високорівневим фреймворком Streamlit, а модуль машинного навчання використовує велику кількість сторонніх бібліотек, ці частини програми вимагають коректного подання методів та структур. Патерн адаптер вирішує проблему несумісності інтерфейсів. Він трансформує об'єктно-орієнтовані структури з мови Python та динамічних таблиць Pandas у специфічний тестовий формат IDF-маніфесту, оскільки це єдина зрозуміла для EnergyPlus форма.

Процес взаємодії розділений на кілька етапів. Перший з них генерація конфігураційного коду. На основі об'єкта, заданого через GUI користувачем, який повністю описує приміщення, IDF-міст формує конфігураційний кодовий маніфест для EnergyPlus. Дані з графічного інтерфейсу зберігаються в форматі json, задля легкості зберігання та транспортування. Задача мосту перевести ці дані на зрозумілу для обчислювального ядра мову, впроваджуючи паралельно графіки інтенсивності (Schedule:Compact) та скрипти Energy Management System. Такий підхід дозволяє зберегти динамічність симуляцій не перенавантажуючи при цьому інтерфейс користувача.

Другий етап оркестрація виконання. Всередині модуля `simulation_engine.py` відбувається синхронізація та менеджмент усіх процесів пов'язаних з

обчислювальним ядром. Так наприклад за допомогою механізму `subprocess.run` запускається `energyplus.exe` як окремий процес у фоновому режимі. Це дозволяє зручніше працювати з ядром, передавати йому необхідні дані, слідкувати за прогресом та обробляти помилки `eplusout.err` не впливаючи тим самим на решту програмного забезпечення. Такий підхід робить систему стабільнішою, витривалішою та передбачуванішою.

Третім етапом є екстракція та трансформація даних. Після завершення ітерації симуляції модуль `simulation_engine.py` виконує роль парсера. Оскільки EnergyPlus виводить дані у файл `eplusout.csv` в своєму форматуванні, важливо перевести їх в зручний для аналізу вигляд та зберегти в систематизованому каталозі. Таким чином по закінченню симуляції модуль починає нормалізувати та каталогізувати дані, роблячи їх зручними та зрозумілими як для розробника, так і для ІІІ-агента.

Такий архітектурний підхід дозволяє ізолювати ядро симуляції від решти застосунку, таким чином вберігаючи загальний проєкт від помилок чи збоїв симуляції, а сам EnergyPlus від впливу на нього інтерфейсів та решти модулів проєкту. В кінці-кінців, таким чином користувач взаємодіє лише з абстракціями Streamlit, а всі важливі та складні обрахунки та інтеграції залишаються за лаштунками, що значно спрощує процес підготовки даних та керування життєвим циклом програми.

Висновки до розділу 2

У результаті розробки структури та обґрунтування компонентів програмного комплексу було розроблено концепцію цифрового двійника інженерної лабораторії, що базується на тривимірних геометричних та теплофізичних об'єктах описаних низькорівневими фескрипторами. Це дозволило на початковому етапі врахувати специфічну Г-подібну структуру приміщення та нівелювати проблему низької теплової інерції огорожувальних конструкцій. Динамічність симуляції

забезпечено гнучким часовим моделюванням антропогенних та технологічних чинників. Спроектована модульна архітектура інтеграційного елементу на базі патернів фасаду та адаптера. Це дозволило повністю ізолювати обчислювальне ядро механізмами фонових підпроцесів та забезпечити надійну обробку помилок. Сформовано захищене масштабоване середовище для подальшої надбудови аналітичних модулів та машинного навчання.

3 Програмна реалізація комплексу HIVANALYZER

3.1 Організація кодової бази та структура каталогу проєкту

Розробка програмного комплексу здійснювалася із дотриманням усіх сучасних стандартів будови модульного ПЗ. Такий підхід включає в себе вимогу до забезпечення низької взаємозв'язності та високого зачеплення окремих частин додатку. Принципи модульності в структуризації репозиторію та організації вихідного коду дозволяють отримати продукт, де кожен архітектурний рівень системи представлений окремим пакетом або каталогом. Це значно спрощує процес пошуку помилок, автоматичної верифікації та подальшого розширення чи супроводу кодової бази додатку. Для чіткого визначення меж системи та детальної специфікації функціональних вимог до системи з боку різних користувачів було розроблено діаграму прецедентів UML (рис. 3.1). Вона допомогла представити загальну логіку взаємодії акторів із ключовими сервісами та внутрішніми аналітичними модулями системи.

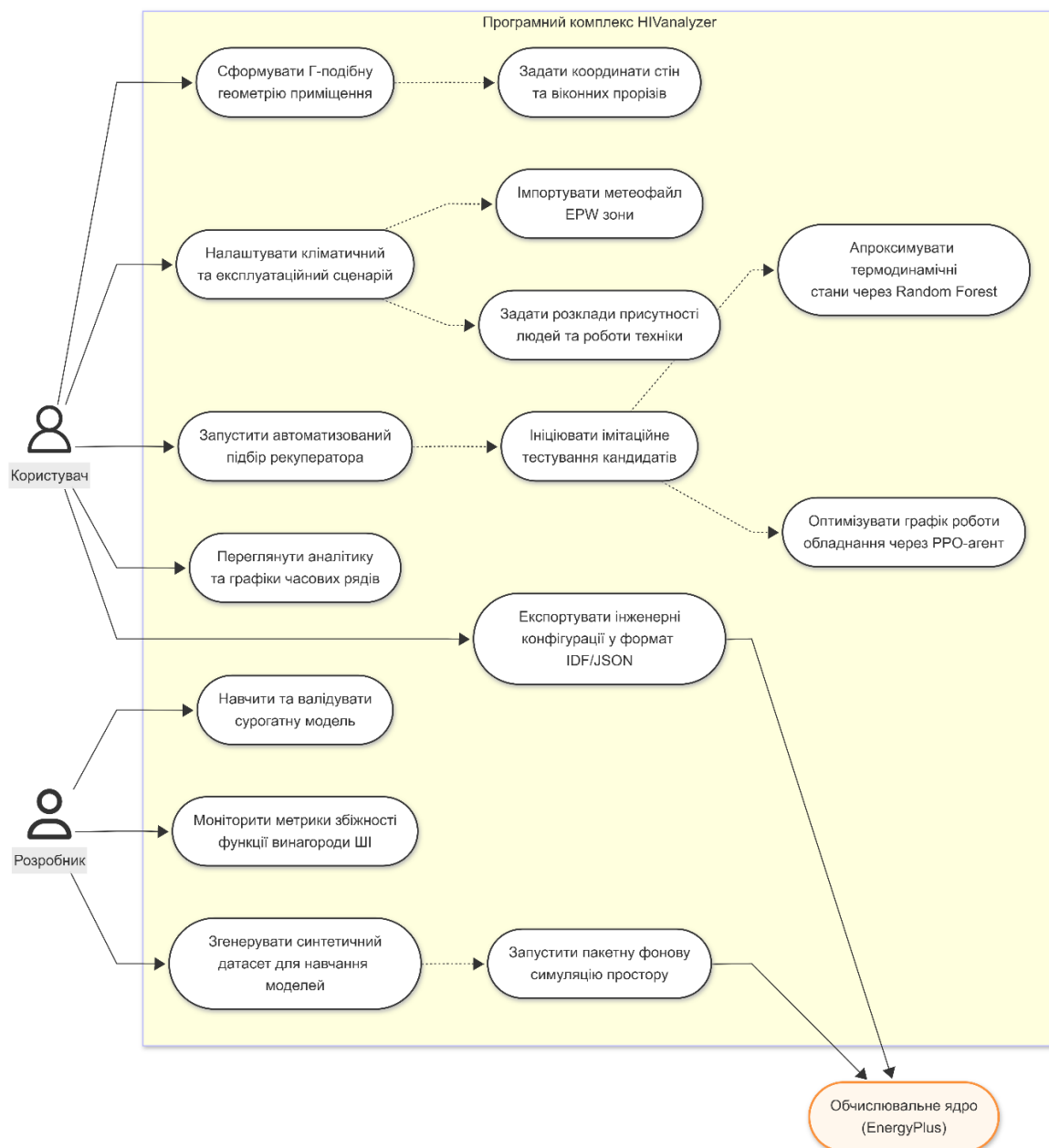


Рисунок 3.1 – Use case діаграма

В ході проєктування було визначено два основні вектори взаємодії відповідно до ролей акторів. Окремі вектори дають конкретному користувачеві різні функціональні можливості. Перший вектор орієнтований на типового користувача. Вважається що він працює з уже налаштованим додатком, тож йому пропонується функціонал конструктора приміщення, функціонал автоматизованого підбору рекуператора та функціонал симуляції роботи обладнання як з підбраною моделлю, так і без неї. Другий вектор орієнтований на розробника, або адміністратора. Його роль полягає в налаштуванні додатку після

встановлення та донавчання моделей машинного навчання за умов виникнення такої необхідності. Йому пропонується функціонал генерування датасету, функціонал налаштування сурогатної моделі для забезпечення високої точності передбачень та функціонал навчання РРО моделі для забезпечення якісного підбору правильного обладнання. Такий підхід до розмежування функціоналу дозволив вирішити проблему перенавантаження інтерфейсу на етапі планування та вберегти користувача від помилкового збиття налаштувань системи.

Для реалізації зазначених прецедентів та координації потоків даних в умовах високої модульності додатку з відокремленим обчислювальним ядром EnergyPlus була розроблена компонентна архітектура. Взаємозв'язки між основними програмними шарами системи показано в діаграмі компонентів (рис. 3.2.).

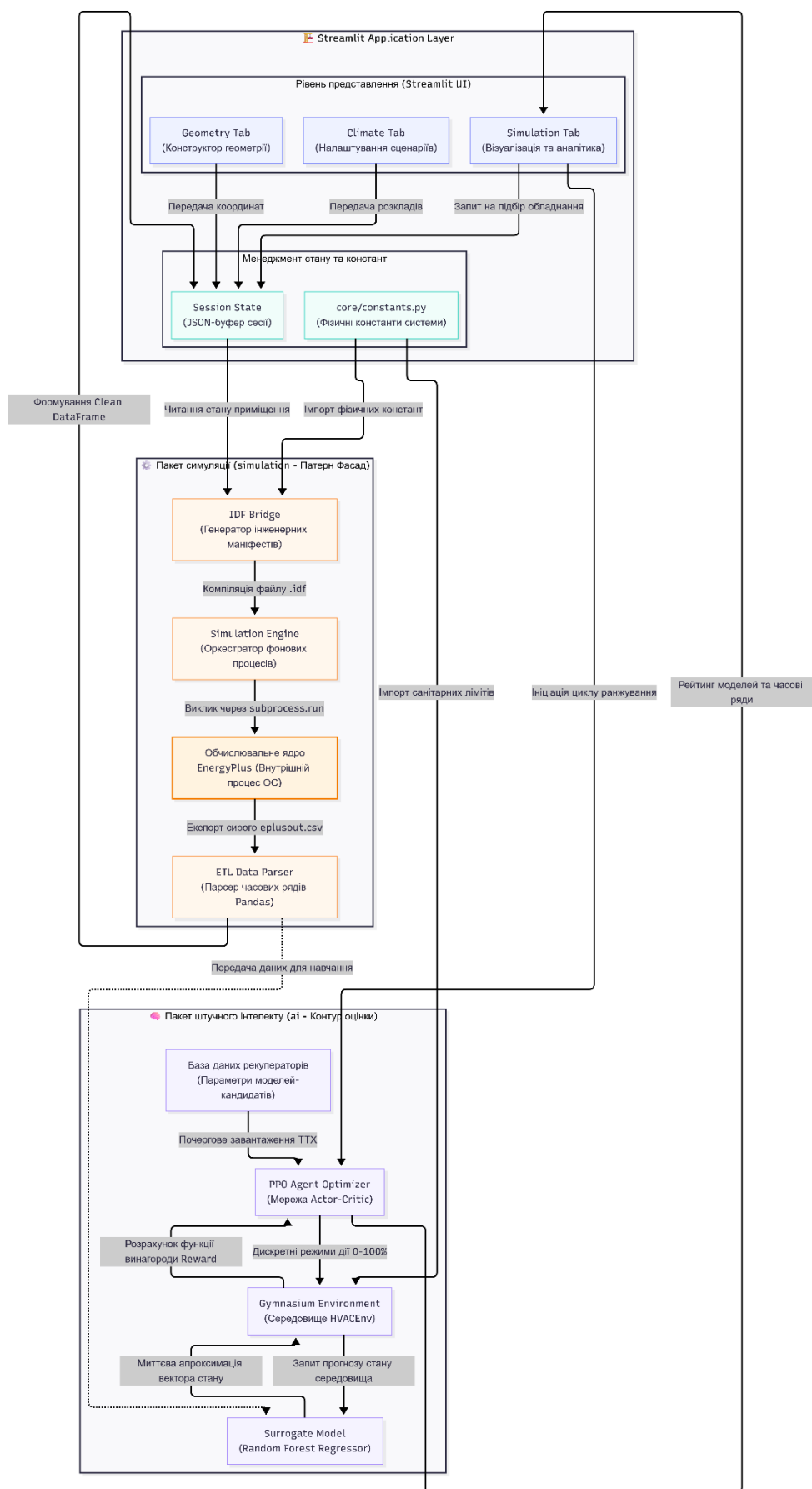


Рисунок 3.2 – Компонента діаграма

Відповідно до архітектурної схеми рівень відображення Streamlit повністю відокремлений від обчислювальних підсистем за допомогою проміжного шару менеджменту сесії. Пакет симуляції, реалізований за допомогою патерну фасаду, інкапсулює у собі взаємодію із зовнішнім консольним ядром EnergyPlus, передаючи очищені дані до пакету штучного інтелекту у вигляді структурованих масивів.

Спроектowana архітектура визначила структуру репозиторію проєкту. Такою кодова база розділена на 6 функціональних пакетів: `core`, `simulation`, `ai`, `ui`, `data` та `tests`. Каталог `core` призначений для збереження глобальних констант та конфігураційних параметрів системи. Модуль `constants.py` містить у собі незмінні фізичні величини, такі як теплоємність повітря чи коефіцієнти теплопередачі різних матеріалів стін, а також нормативні екологічні константи, як, наприклад, критичну межу вмісту вуглекислого газу в 1000ppm. Централізація цих даних допомагає уникнути дублювання даних в різних модулях та забезпечити єдине джерело істини для обчислювального та нейромережевого контурів.

Пакет `simulation` є серцем інженерної складової цифрового двійника. Модуль `idf_bridge.py` відповідає за динамічне конструювання текстових маніфестів для обчислювального рушія. Він трансформує високорівневі параметри приміщення отримані з графічного інтерфейсу в низькорівневі об'єкти EnergyPlus, додає конфігураційні параметри симуляції, часові рамки та правила поведінки. Модуль `simulation_engine.py` реалізує патерн фасаду. Його головна місія – управління зовнішнім консольним додатком `energyplus.exe`. Взаємодія з ним організована через викликання підпроцесів операційної системи за допомогою бібліотеки `subprocess`. Модуль контролює життєвий цикл процесу, отримує результати симуляції та логує помилки. Після успішного завершення процесу `simulation_engine` також відіграє роль ETL-компонента. Модуль витягує згенеровані дані, очищує їх від зайвої інформації, нормалізує та зберігає структуровані `DataFrame` об'єкти бібліотеки `Pandas`. Останнім модулем пакету симуляції є `validation.py`. Цей модуль аналізує результати симуляції, шукає

відхилення від норми та оцінює дотримання конкретних вимог. Таким чином можна прослідкувати як помилки симуляції, що можуть виникнути через некоректне конструювання приміщення в графічному інтерфейсі, так і оцінити наскільки якісно симульовані прилади виконали свою роботу.

Пакет `ai` поєднує в собі інтелектуальну логіку обробки, прогнозування та ухвалення рішень. Модуль `rl_environment.py` задає середовище навчання створене за допомогою інструментарію `Gymnasium`. Воно виконує роль сполучної ланки між програмними модулями та ШІ-агентом, допомагаючи створити логіку кроків симуляції, визначення функціональної корисності та розрахувати функцію винагород. Модуль `ml_surrogate.py` навчає та зберігає сурогатну модель, що імітує реальний обчислювальний блок за допомогою алгоритму `LightGBM` [10]. Модуль згодовує агенту реальні результати симуляцій та виступає для нього вчителем, аби навчена модель могла відслідковувати та точно повторювати нелінійні залежності чинників впливу на температурні показники та значення якості повітря всередині приміщення. Модуль `rl_agent.py` містить в собі логіку основної моделі прийняття рішень. За допомогою навчання з підкріпленням та методів `PPO` модуль навчає агента, що може аналізувати процеси, які відбуваються в приміщенні, та з огляду на них методом підбору визначити обладнання, що найкраще підходить конкретній кімнаті. Модуль `ai_engine.py` відповідає за оркестрацію всіх процесів навчання. Саме він передає необхідні дані середовищу навчання, зберігає моделі та підвантажує навчені агенти для роботи.

Пакет `ui` відповідає за графічний інтерфейс застосунку на базі фреймворку `Streamlit`. Модуль `designer` задає загальний стиль застосунку. Тут зберігаються ті підтягуються всі графічні елементи, кольори, шрифти, а також задаються геометричні форми деяких об'єктів інтерфейсу. Кожна вкладка графічного інтерфейсу описана окремим модулем. Кожен такий модуль окремо рендерить необхідні елементи для конкретної вкладки а також підтягує сторонні графічні елементи, такі як `Plotly` [11] лише за виникнення потреби, таким чином економлячи ресурси комп'ютера скорочуючи загальний час підвантаження додатку.

Пакет tests відповідає за автоматизоване тестування додатку. Пакет містить набір окремих модулів, побудованих на базі PyTest [12], кожен з яких відповідає за окрему функціональну частину. Разом вони гарантують стабільність системи, виявляють вразливості чи схильності до поломок та контролюють якість інженерних даних та роботу ІІІ-агентів.

3.2 Програмна реалізація пакета координування симуляцій

Пакет simulation є фундаментальним інженерним прошарком програмного комплексу. Його головне завдання полягає в ізоляції та автоматизації процесів взаємодії між прикладним інтерфейсом Streamlit та зовнішнім консольним застосунком термодинамічного обчислювального ядра EnergyPlus. Реалізація пакеті полягає в застосування патернів фасаду та адаптера. Це дозволяє заховати складні низькорівневі маніпуляції з файлами конфігурацій та маніфестами, запуск паралельних підпроцесів та парсинг сирих інженерних звітів за декількома високорівневими функціями програмного інтерфейсу.

Функціональна структура пакета розділена між трьома взаємопов'язаними модулями: `idf_bridge.py`, `simulation_engine.py` та `validator.py`. Кожен з них відіграє певну роль в загальному конвеєрі виконання інженерних експериментів, генерації синтетичних даних чи верифікації існуючих.

Модуль `idf_bringe.py` відповідає за переведення високорівневих даних, що надходять у форматі `json`, у текстові маніфести специфічного формату `Input Data File`, призначеного для виконання симулятором. Головною точкою входу модуля є функція `generate_idf_structure(project_json: str) -> str`. Вона десереалізує вхідний рядок `json`, обчислює базові просторові метрики та послідовно викликає решту приватних підфункцій, які формують решту окремих секцій маніфесту симуляції.

Оскільки ядро EnergyPlus написане мовами C та Fortran, воно є вразливим до використання сторонніх символів та кирилиці. Для уникнення збоїв кодування та завершення програми було розроблено функцію `transliterate(text: str) -> str`. Вона

оперує словником мапування та автоматично переводить кириличні знаки в латинку, видаляє сторонні символи та замінює знаки пробілу нижнім підкресленням, автоматично забезпечуючи сумісність з симулятором на рівні імен об'єктів.

Функція `_generate_global_objects` формує загальносистемні параметри обчислювального експерименту. Вона записує дескриптори `SimulationControl`, встановлює крок дискретизації часу `Timestep` рівним 4, що означає проведення розрахунків кожні 15 хвилин, ініціалізує ліміти типів розкладів та активує обчислення балансу забруднювачів у зоні через об'єкт `ZoneAirContaminantBalance`. Рівень зовнішнього фонового вуглекислого газу динамічно підставляється із фізичних констант ядра. Також функція генерує географічні координати об'єкта `Site:Location` для міста Києва (50.45 північної широти, 30.52 східної довготи) та детермінує річний період розрахунку `RunPeriod` на весь 2026 рік.

Програмний блок `_generate_materials_and_constructions` відповідає за опис теплофізичних властивостей огорожувальних конструкцій. Функція трансліює обраний користувачем матеріал стін у відповідні англомовні дескриптори та зчитує показники теплопровідності, щільності та питомої теплоємності з глобального словника `constants.MATERIALS`. На основі цих даних та товщини стін динамічно конструюється багатошаровий пиріг конструкції `Construction`, що включає штукатурний, ізоляційний та несучий шари, а також задає параметри спрощеного скління через об'єкт `WindowMaterial:SimpleGlazingSystem` із фіксованим коефіцієнтом теплопередачі рівним 2.7 та коефіцієнтом сонячного теплонадходження на рівні 0.7.

Моделювання геометрії кімнати реалізовано у функції `_generate_geometry`. Програма підтримує автоматичний розрахунок як стандартних прямокутних кімнат, так і складних нетипових Г-подібних топологій лабораторій. Функція зчитує довжину, ширину, висоту та лінійні параметри кутового вирізу (L_{cut} , W_{cut}). Об'єм зони обчислюється автоматично з урахуванням геометрії: для Г-подібної форми від площі загального прямокутника віднімається площа вирізу, після чого

значення множиться на висоту. Для побудови стін через об'єкти `BuildingSurface:Detailed` застосовано алгоритм циклічного обходу вершин на основі масивів координат. Для забезпечення правильного розрахунку теплопровідності та радіаційного обміну всі вектори нормалей поверхонь повинні бути спрямовані назовні кімнати. Тому для стін та даху вершини записуються проти годинникової стрілки, а для підлоги напрямком генерації масиву координат автоматично реверсується за годинниковою стрілкою.

Функція `_generate_windows` реалізує оригінальний алгоритм просторового координатного зіставлення для динамічного вбудовування вікон у раніше згенеровані стіни через об'єкт `FenestrationSurface:Detailed`. Оскільки інтерфейс користувача передає вікна як плоскі об'єкти з лінійними координатами X та Y , функція розраховує математичну відстань від вікна до кожної з наявних стін будівлі. Використовуючи просторовий допуск епсилон рівний 0.1 метра, алгоритм визначає приналежність вікна (наприклад, якщо Y близьке до нуля, вікно належить до стіни `Bottom_Wall`), після чого розгортає двовимірні координати у тривимірні вершини меж скління з фіксованою висотою підвіконня 0.9 метра та верхньою межею 2.1 метра.

Моделювання внутрішніх теплових та екологічних навантажень виконується у функції `_generate_schedules_and_internal_gains`. Модуль підтримує три профілі експлуатації: «Офіс», «Житлове», «Серверна», конструюючи погодинні розклади присутності людей та виділення тепла обладнанням. Для моделювання нелінійних сплесків забруднення, характерних для інженерних лабораторій, в код маніфесту вбудовується підсистема макросів `EnergyManagementSystem`. Програма створює EMS-алгоритм `Random_Soldering_Logic`. На початку кожної доби алгоритм генерує випадкову величину за функцією `@RandomUniform`; якщо значення перевищує поріг 0.7, активується актуатор `Spike_Actuator`, який у проміжку часу між 15:00 та 16:00 годинами штучно ініціює інтенсивні викиди дрібнодисперсного пилу через об'єкти `ZoneContaminantSourceAndSink:Generic:Constant`. Потужність виділення шкідливих речовин та виділення тепла від 3D-принтерів динамічно масштабується

за допомогою коефіцієнтів активації, які надходять із графічного інтерфейсу проекту.

Програмне керування вентиляцією реалізовано у функції `_generate_hvac_and_ventilation`. Вона інтегрує об'єкт рекуператора `ZoneHVAC:EnergyRecoveryVentilator` та теплообмінник `HeatExchanger:AirToAir:SensibleAndLatent`, параметри яких зчитуються з технічного паспорта обраної моделі обладнання. Для забезпечення динамічного регулювання повітрообміну в код вбудовується другий EMS-контролер `DCV_Control`, що реалізує логіку вентиляції за потребою. Написаний код макросу щохвилинно зчитує показники датчиків вуглекислого газу та пилу, обчислює необхідну швидкість роботи припливного та витяжного вентиляторів `Fan:SystemModel` за допомогою нелінійної кубічної кривої потужності `Curve:Cubic` та передає масовий потік повітря на відповідні системні актуатори, утримуючи параметри середовища в межах норми. Додатково функція ініціалізує уставки термостата та резервну систему кондиціонування й опалення `ZoneHVAC:IdealLoadsAirSystem` із чітким обмеженням максимальної потужності електричного нагрівача.

Модуль `simulation_engine.py` виконує функцію оркестратора обчислювальних процесів операційної системи та забезпечує повний цикл автоматизації інженерного моделювання. Головна функція `run_simulation` здійснює фоновий запуск консольної утиліти `energyplus.exe` через системний інструмент `subprocess.run` в ізольованих робочих директоріях, що повністю запобігає конфліктам спільного доступу до файлів. Після завершення обчислень викликається функція `get_results`, яка реалізує ETL-пайплайн трансформації вихідних даних за допомогою бібліотеки `Pandas`: вона автоматично мапує складні назви системних колонок у короткі інженерні токени (`T_in`, `CO2`, `Fan Energy`, `Heating Energy`) та коригує нетиповий формат часових міток симулятора, замінюючи значення `24:00:00` на `00:00:00` із відповідним календарним зсувом дат.

Для автоматизованого формування масивів даних, необхідних для побудови сурогатних моделей штучного інтелекту, у модулі реалізовано керуючий клас `SimulationRunner`. Його метод `run_batch` у циклі запускає інженерні експерименти на основі випадкової мутації фізичних параметрів приміщення та вибору одного з п'яти експлуатаційних стрес-сценаріїв. Отримані часові ряди проходять через конвеєр інженерії ознак, де обчислюються календарні атрибути та лагові змінні із запізненням до двох годин для врахування теплової інерції огорожувальних конструкцій, після чого фінальна матриця ознак фіксується і вертикально записується у файл `training_dataset.csv`.

Завершальним етапом обробки інженерного контуру є робота аналітичного модуля `validator.py`, який реалізує концепцію автоматизованої валідації результатів на відповідність фундаментальним фізичним законам. Створений клас `ModelValidator` виконує вісім спеціалізованих інженерних тестів, що оцінюють темп остигання кімнати, піки та нічне відновлення газів, кореляцію опалення з погодою, а також розраховують фактичний ККД рекуператора і швидкість очищення повітря від пилу. Головний метод класу `run_all_tests` здійснює пакетний запуск усіх перевірок, агрегує отримані статуси (Pass, Warning, Fail) і серіалізує фінальний аналітичний звіт у JSON-лог для його подальшої візуалізації на рівні графічного інтерфейсу користувача.

3.3 Реалізація обчислювального ML-контуру програми

Розробка інтелектуальної системи підбору та керування обладнанням базується на алгоритмах навчання з підкріпленням. Для такої методики ключовим моментом стає створення правильного середовища та зв'язок з реакційним цифровим двійником. Аби забезпечити сумісність з більшістю сучасних алгоритмів ШІ було прийнято рішення використовувати фреймворк `Gymnasium` (розвиток бібліотеки `OpenAI Gym`).

Програмна реалізація середовища базується в класі HVACEnv, який унаслідок клас gym.Env. Цей модуль виконує роль інкапсуляції фізичної логіки в стандартизований API, що дозволить будь-якому RL-агенту легко взаємодіяти з цифровим двійником приміщення без необхідності розуміти внутрішню інфраструктуру EnergyPlus.

Архітектура середовища характеризується ітераційним циклом взаємодії, що можна розбити на наступні ключові методи. `__init__`, на етапі ініціалізації система визначає простори спостережень (`observation_space`) та дій (`action_space`). Система конфігурує межі нормалізації параметрів мікроклімату та підвантажує доступні технічні характеристики моделей з бази даних. `reset()`, цей ініціалізує початок нового епізоду навчання моделі. Ним виставляються початкові умови температури, концентрації CO₂ та PM, часу доби та повертається агенту перший вектор стану. Таким чином навчання агента можна починати з різних сценаріїв. `step(action)` – це центральний метод, який реалізує логіку переходу системи від одного стану в інший. Агент передає методу обрану дію, після чого середовище обраховує зміни в симуляції, визначає їх наслідки, формує винагороду/покарання та передає усе це назад агенту у вигляді кортежу даних. Всі ці процеси наочно продемонстровано блок-схемою на рисунку 3.3.

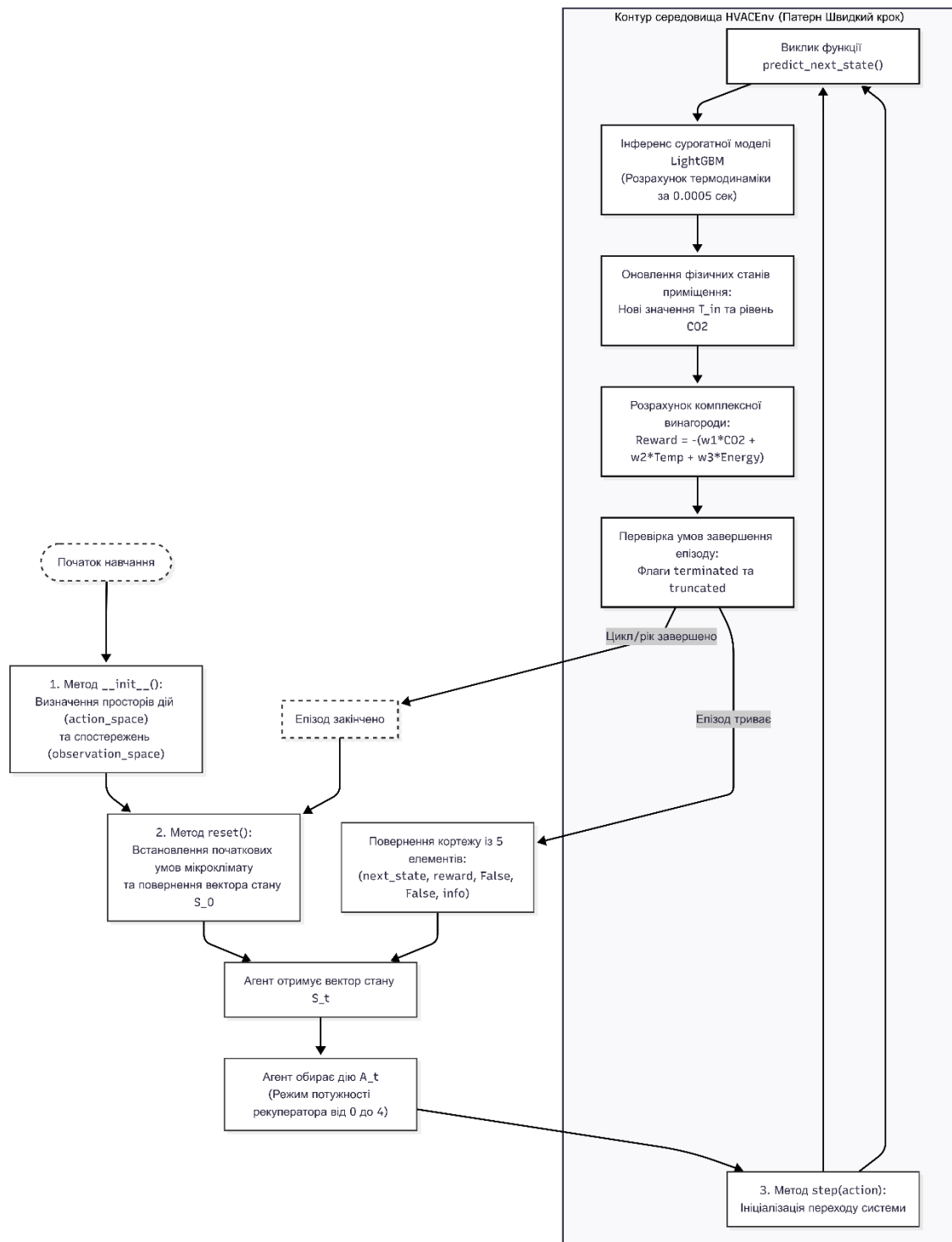


Рисунок 3.3 – Блок-схема ітераційного циклу взаємодії агента з середовищем

Для забезпечення відповідності вимог реалістичності інтелектуального програмного забезпечення, програмний інтерфейс був деталізований на рівні специфікації станів та функції винагород. Таким чином простір дій було сформовано у вигляді дискретного набору типів `gymnasium.spaces.Discrete(5)`, де

кожне число від 0 до 4 мапується на фіксовані режими інтенсивності роботи системи вентиляції: 0%, 25%, 50%, 75% та 100% від максимальної паспортної потужності інтенсивності моделі. Це дозволило реалістично задати режимову поведінку обладнання та не допустити застрягання інтелектуальної моделі на неможливому рівні інтенсивності, навіть якщо він є найоптимальнішим для конкретних умов.

Простір спостережень `observation_space` реалізовано через структуру `gymnasium.spaces.Box`, яка визначає межі багатовимірному вектора неперервних величин. До складу цього вектора входять: поточна внутрішня температура в приміщенні, зовнішня температура повітря, рівень концентрації вуглекислого газу в зоні, поточний час доби, а також набір попередніх виборів, сформованих інженерним конвеєром. Всі вхідні елементи вектора станів проходять обов'язкову процедуру мін-макс нормалізації [13] для забезпечення стабільності збіжності градієнтних методів оптимізації нейромережових моделей III.

Центральний метод `step(action)` містить в собі алгоритм комплексного розрахунку формули винагород. Функція складається як з винагород, так і зі штрафів. Агент отримує фіксовану винагороду за утримання кімнати в оптимальних умовах, що дає лінійне зростання балу агента, коли той успішно виконує свою роботу. Штрафування вираховується сумою з трьох вагових метрик: якість повітря, температура повітря та кількість спожитої енергії. Для інженерної лабораторії найбільш пріоритетною метрикою є якість повітря, слідом іде температура, що залишає енергоефективність останньою. Математична логіка обчислення балу для кожного кроку реалізована за формулою 3.1.

$$R = -(w_P \cdot P_{penalty} + w_C \cdot C_{penalty} + w_T \cdot T_{penalty} + w_E \cdot E_{penalty}) + S_{bonus} \quad (3.1)$$

Де R – бал математичної винагороди, $P_{penalty}$, $C_{penalty}$, $T_{penalty}$ та $E_{penalty}$ – експозиційні штрафи, w_P , w_C , w_T , w_E – ваги штрафів за порушення оптимальних показників рівня забруднення дрібнодисперсним пилом, кількості вуглекислого газу, відхилення від оптимальної температури та витрату електроенергії відповідно,

а S_{bonus} – це фіксований бонус за кожен крок проведений в межах нормативних показників.

Використання рушія EnergyPlus як середовище для безпосереднього навчання RL-агента залишає один суттєвий ботлнек, а саме тривалість кожної ітерації. Коли один крок такої симуляції займає від 1 до 3 секунд, глибоке навчання на мільйони ітерацій стає абсолютно непрактичним. Для розв’язання цієї проблеми було реалізовано додаткову модель машинного навчання за принципами сурогатності.

Сурогатна модель – це апроксиматор, який вчиться імітувати складну математичну симуляцію та давати результати максимально подібні до неї, але в тисячі разів швидше.

В процесі розробки сурогатної моделі було пройдено наступні етапи:

Генерація навчального набору даних. В інтерфейсі проєкту є спеціалізовані вкладки для розробників. Одна з них `data_gen_tab.py` дозволяє розробнику запустити велику кількість паралельних симуляцій в EnergyPlus. Скрипт розроблений таким чином щоб запустити та зберегти велику кількість симуляцій з різними сценаріями та вхідними даними, фіксуючи реакцію обчислювального ядра на різні обставини. Таким чином створюється не просто обширний датасет для навчання, а повноцінний «зліпок» поведінки EnergyPlus.

Вибір алгоритму та архітектури моделі. Для апроксимації табличних даних було вибрано ансамблеві методи навчання, зокрема алгоритм XGBoost (Extreme Gradient Boost). Такий вибір базується на тому, що ця модель здатна ефективно обробляти нелінійні залежності між термодинамічними показниками та якістю повітря. Модель налаштована для аналізу наступного вектора стану (формула 3.2):

$$S_{t+1} = f(S_t, A_t) \quad (3.2)$$

де S_t – це поточні стани температури, CO₂ та PM, а A_t – це дія агента, що характеризується моделлю рекуператора та його потужністю.

Таким чином було розроблено конвеєр підготовки даних та навчання сурогатної моделі апроксимації, структуру якого наочно показує рисунок 3.4.

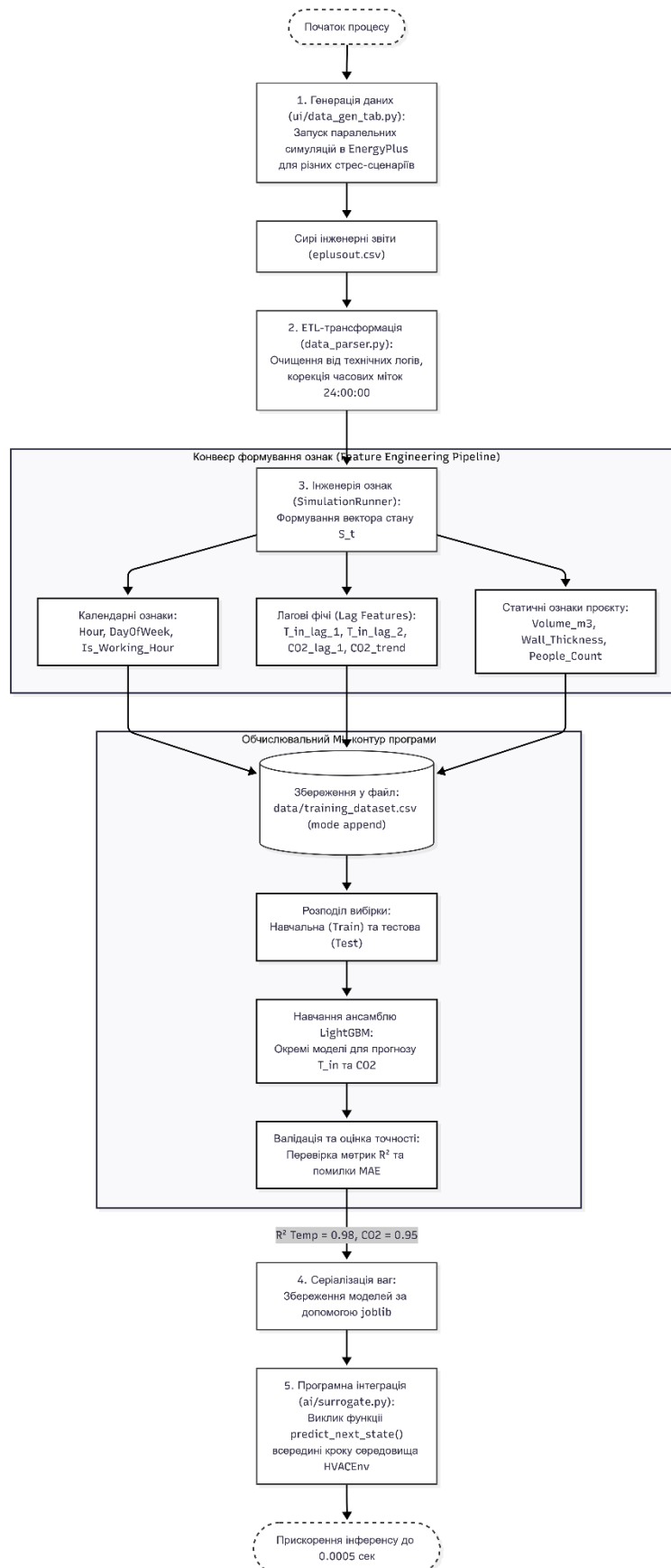


Рисунок 3.4 – Діаграма конвеєра підготовки даних та навчання моделі

У ході програмної оптимізації обчислювального контуру комплексу було проведено порівняльний аналіз архітектур ансамблевого навчання. Хоча на етапі прототипування досліджувався алгоритм XGBoost, для фінальної реалізації сурогатного рушія у модулі `ml_surrogate.py` було обрано фреймворк LightGBM. Таке інженерне рішення зумовлене специфікою розгортання інтелектуальних систем на користувацьких ЕОМ: LightGBM використовує алгоритм побудови дерев на основі гістограм та оптимізований підхід вертикального розростання, що забезпечує значно менше споживання оперативної пам'яті та критично вищу швидкість обробки запиту у порівнянні з класичним XGBoost чи Random Forest.

Програмний код інтеграції сурогатної моделі інкапсульовано у функцію `predict_next_state(current_observation, action_params)`. Завантаження навчених ваг моделей LightGBM для прогнозування температури та концентрації CO₂ здійснюється один раз під час ініціалізації системи за допомогою бібліотеки `joblib` або вбудованих методів серіалізації LightGBM, що виключає дискові затримки при виконанні ітерацій навчання. Функція приймає вектори поточного стану, виконує їх трансформацію, додаючи лагові ознаки, сформовані модулем оркестрації симуляцій, та об'єднує їх із вектором параметрів дії. Модель повертає точні апроксимовані значення фізичного стану приміщення на наступний часовий крок.

Програмна інтеграція та оцінка точності. Ця модель інтегрована в Gynasium-середовище через функцію `predict_next_state()`. Валідація на тестовій вибірці показала досить високу точність як для імітації настільки складної та нелінійної симуляції. Коефіцієнт детермінації R^2 показав значення 0.98 для температури та 0.95 для забруднення повітря. Але головним досягненням моделі є скорочення середнього часу ітерації навчання від 2 секунд, за умови використання EnergyPlus, до 0.0005 секунди за умови використання сурогатної моделі. Таким чином навчання RL-моделі для підбору оптимального обладнання може відбуватися на користувацькому комп'ютері в розумні терміни, нівелюючи тим самим необхідність в потужному сервері для розгортання програмного забезпечення.

3.4 Програмна реалізація логіки вибору моделей та алгоритму PPO

Завершальним етапом розробки інтелектуального модуля є побудова та конфігурація агента. Для оптимального вирішення задачі вибору рекуператора було обрано алгоритм Proximal Policy Optimization (PPO). Даний вибір обґрунтований високою ефективністю та стабільністю використання даних, а також здатність алгоритму працювати як з дискретними так і з безперервними просторами дій.

Програмна реалізація зосереджена в модулях `rl_agent.py` та `train_agent.py`. В основі стеку лежить бібліотека Stable Baselines3 [14], яка використовує фреймворк PyTorch [15] для побудови та оптимізації обчислювальних графів нейронних мереж. У структурі інтелектуального контуру модуль `rl_agent.py` відповідає за побудову логіки та архітектури III-агента, в той час як `train_agent.py` керує процесами запуску, зупинки та конфігурації сесій навчання. Вибір бібліотеки Stable Baseline3 обґрунтований можливістю впровадження стабільної on-policy реалізації для алгоритму PPO, таким чином мінімізувавши архітектурні ризики та запобігши виникненню програмних помилок під час розрахунку градієнтів. Фреймворк PyTorch виступає базовим математичним двигуном, виконуючи швидкі матричні обчислення та автоматичне диференціювання обчислювальних графів без надмірного навантаження на апаратні ресурси користувацької ЕОМ.

Процес побудови та навчання агента включав наступні етапи:

Архітектура нейронної мережі. Для узагальнення стратегії та функції цінності було використано багатошаровий перцептрон. Мережа отримує із середовища нормалізований вектор стану та повертає ймовірнісний розподіл дій відповідно до кожного режиму роботи рекуператора.

Конфігурація гіперпараметрів. Під час розробки та тестування моделі, імперичним шляхом було визначено набір оптимальних параметрів, таких як

швидкість навчання, обсяг пакету даних та коефіцієнт дисконтування. Усі ці параметри було збережено в модулі `train_agent.py`.

Моніторинг навчання. В процесі налаштування параметрів виникла потреба моніторингу навчання в реальному часі. Задля візуалізації процесу оптимізації та контролю збіжності алгоритму було використано інструментарій TensorBoard. Це дозволило відслідковувати криву середньої винагороди та виявляти ефекти перенавчання або деградації стратегії.

Експорт моделі. Плато на графіку винагород означає, що агент досяг своїх оптимальних параметрів, в такому випадку прогрес навчання потрібно зберегти для подальшого використання. Модель зберігається у форматі `.zip` у відповідній директорії всередині проєкту. Надалі модуль `ai_engine.py` відповідає за кешування моделі в оперативну пам'ять та використання її у вкладці аналітики користувацького інтерфейсу.

Результатом навчання став набір патернів поведінки нейронної мережі, які дозволяють їх в режимі реального часу знаходити закономірності між роботою приладів, антропогенним навантаженням та змінами температури і якості повітря. Агент показує проактивну поведінку, наближену до реальних розумних модулів керування мікрокліматичним обладнанням, які спрямовані на забезпечення стабільності мікроклімату. Таким чином програмний комплекс трансформує теоретичні знання про роботу рекуператорів у повноцінну автономну систему прийняття рішень.

3.5 Програмний код інтерактивного Streamlit UI

Програмна реалізація прикладного рівня програмного комплексу розроблена за допомогою фреймворку Streamlit, що підтримує принципи реактивного декларативного програмування. Вхідною точкою до інтерфейсу є модуль `main.py`. Він виконує первинну конфігурацію сервера, підвантажує необхідні стилі та параметри оформлення та будує верхній рівень навігаційної панелі застосунку.

Задля оптимізації архітектури графічна оболонка розділена на окремі модулі, кожен з яких відповідає за свою вкладку в інтерфейсі: `geometry_tab.py`, `advisor_tab.py`, `ai_lab_tab.py`, `save_tab.py`, `climate_tab.py`, `data_gen_tab.py`. Стилями та матеріалами застосунку керує окремий модуль `designer.py`, що дало можливість побудувати уніфіковану стильову мову застосунку для усіх вкладок інтерфейсу та забезпечити стабільність рендерингу графічних матеріалів.

Синхронізація інформаційних потоків між ізольованими компонентами вкладок забезпечується централізованим сховищем оперативної пам'яті фреймворку `st.session_state`. Під час первинного запуску програми функція `designer.init_session_state` виконує декларативний опис та заповнення обмінних структур даних конфігураційними дефолтами із файлу `data/default.json`. Стан сесії перманентно утримує два основні масиви даних: словник налаштувань проекту `st.session_state.project_settings` та структуровану таблицю доданих інженерних елементів кімнати `st.session_state.elements_df`. Вкладка проектування геометрії `geometry_tab.py` зчитує ці масиви та рендерить двовимірний ескізний план приміщення засобами графічної бібліотеки `Plotly` через метод `designer.render_blueprint`, відображаючи типи стін та точне просторове розташування встановлених об'єктів. Швидка модифікація геометричних координат, габаритів та теплової потужності джерел реалізована через інтерактивний табличний редактор `st.data_editor`, зміни в якому миттєво перезаписуються в пам'ять сесії. Експорт та збереження поточного стану двійника організовано у модулі `save_tab.py`, де підключено кнопки скачування серіалізованого JSON-файлу проекту та згенерованого інженерного маніфесту IDF, а також розгорнуто текстову область для попереднього перегляду вихідного коду конфігурації симулятора.

Прикладний інженерний аналіз та взаємодія з навченими нейромережами здійснюються через модулі `climate_tab.py` та `advisor_tab.py`. У вкладці точного розрахунку `climate_tab.py` реалізовано інтерактивні слайдери уставки термостатів, блоки математичного розрахунку погодинної зміни концентрації вуглекислого газу

та автоматизований підбір рекуператора методами III. Після натискання кнопки запуску двійника функція `_run_energyplus_simulation` транслює параметри пам'яті в IDF-структуру, ініціює фоновий розрахунок, парсить результати і відображає три вкладки деталізованих графіків Plotly (Температура, CO₂, Якість повітря) з підтримкою повзунків масштабування (Rangeslider). Очищені ряди Pandas DataFrame автоматично передаються об'єкту `validator.ModelValidator()`, який виводить розгорнутий звіт оцінки реалістичності фізичних процесів та будує полярну радарну діаграму загального індексу достовірності моделі. Вкладка інтелектуального радника `advisor_tab.py` завантажує PPO-модель через `ai_engine.load_ai_model`, у фоновому режимі виконує паралельну добову симуляцію ручного та автоматичного управління, виводить порівняльні метрики енергоефективності з розрахунком відсотка економії та формує логічне Success-текстове пояснення для користувача за допомогою алгоритмів Пояснювального III.

Для проведення науково-дослідних експериментів режим розробника активує вкладки `data_gen_tab.py` та `ai_lab_tab.py`. Модуль генерації даних дозволяє налаштовувати обсяг пакетних запусків EnergyPlus, автоматично кешує результати, вираховує індекс багатства зібраного датасету, будує гістограми розподілу та теплові карти матриць кореляції ознак. Скрипт лабораторії III `ai_lab_tab.py` керує процесами тренування сурогатної фізики LightGBM з виведенням діаграм розсіювання та точок оцінки похибок MAE/RMSE. Найбільш складною архітектурною частиною цього модуля є запуск контуру донавчання PPO-агента. Оскільки Streamlit працює за принципом повного перезапуску скрипта зверху вниз при будь-якій зміні стану, пряме виконання тривалих ітераційних циклів у головному потоці призводить до замерзання інтерфейсу та виникнення WebSocket-помилки синхронізації з браузером. Для усунення цієї проблеми алгоритм навчання виконується всередині ізольованого фонового потоку операційної системи. Взаємодія між фоновим та головним візуалізаційним потоками Streamlit організована через thread-safe словник `shared_state`. Керуючий цикл головного

потоків періодично зчитує з цього словника поточний відсоток прогресу, кількість пройдених таймстепів та масиви отриманих нагород, динамічно оновлюючи смугу завантаження `st.progress` та лінійний графік середньої винагороди в режимі реального часу без ризику обриву клієнтського з'єднання.

Висновки до розділу 3

У третьому розділі повністю описано програмну реалізацію програмного комплексу. На основі інженерних принципів низької взаємозв'язності та високого зачеплення розроблено модульну структуру кодової бази, що дозволило відокремити інженерні розрахунки, алгоритми машинного навчання та інтерфейс користувача. Створено пакет координування симуляцій, який автоматизує процес генерації IDF-маніфестів симуляції, керує фоновією роботою підпроцесу обчислювального ядра `EnergyPlus` та виконує автоматичну трансформацію отриманих результатів у високорівневий `DataFrame` формат з подальшою верифікацією фізико-орієнтованою верифікацією відповідності термодинамічним законам.

Реалізовано обчислювальний контур машинного навчання, що поєнує `Gymnasium`-середовище, сурогатну модель математичної апроксимації побудовану на базі градієнтного бустингу та алгоритм оптимізації за політикою `PPO`. Застосування навченого `LightGBM` ансамблю дозволило замінити часоємкий `EnergyPlus` значно скоротивши крок навчання агента прийняття рішень зробивши реалістичним сценарій донавчання його під конкретне користувацьке середовище застосування.

За допомогою фреймворку `Streamlit` розроблено реактивний графічний інтерфейс користувача, де керування тривалими обчислювальними сесіями навчання ШІ-агентів ізольовано у фоновому потоці через потокобезпечний обмін станами, що виключає завмирання інтерфейсу та гарантує стабільність роботи веб-застосунку.

4 ВЕРИФІКАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ ТА ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Автоматизоване модульне тестування та аналіз складності й підтримуваності вихідного коду

Важливим етапом розробки програмного комплексу є його верифікація забезпечення відповідності стандартам індустрії з точки зору надійності обчислень, якості логіки та відмовостійкості інтелектуальних модулів. Оскільки система поєднує в собі роботу зі стороннім обчислювальним ядром, процеси парсингу та трансформувannya даних і нейромережжеві обрахунки, стратегія тестування була розділена на два підходи: модульне тестування коректності роботи застосунку за допомогою фреймворку PyTest та аналіз складності та підтримуваності коду за допомогою утиліти Radon [16].

Динамічна верифікація системи реалізована через пакет автоматизованих модульних тестів. Було розроблено 34 окремих сценарії тестування, кожен з яких успішно завершився без збоїв чи відхилень від очікуваних результатів. Рисунок 4.1 наочно демонструє перелік тестових сценаріїв, час їх виконання та статус завершення. Загальна швидкість виконання повного тестового пакету склала 49 секунд, що свідчить про високу ефективність відпрацювання математичних заглушок.

Summary			
34 tests took 00:00:49.			
(Un)check the boxes to filter the results			
0 Failed 34 Passed 0 Skipped 0 Expected failures 0 Unexpected passes 0 Errors 0 Retruns 0 Retired			Show all details / Hide all details
Result	Test	Duration	Links
Passed	test/test_at_engine.py::test_generate_rot_explanation	2 ms	
Passed	test/test_at_engine.py::test_simulate_2th_feedback_math	00:00:03	
Passed	test/test_at_env.py::test_reset_env_initialization	4 ms	
Passed	test/test_at_env.py::test_reset_calculation_extreme_cs2	4 ms	
Passed	test/test_at_env.py::test_reward_calculation_extreme_temperature	5 ms	
Passed	test/test_at_env.py::test_normalization_bounds	4 ms	
Passed	test/test_app.py::test_hul_app	00:00:01	
Passed	test/test_core.py::test_materials_constants	1 ms	
Passed	test/test_core.py::test_at_physics_constants	1 ms	
Passed	test/test_core.py::test_simulation_settings	1 ms	
Passed	test/test_designer.py::test_ml_session_state	5 ms	
Passed	test/test_designer.py::test_add_element	9 ms	
Passed	test/test_designer.py::test_export_project	9 ms	
Passed	test/test_extract.py::test_extract_results	35 ms	
Passed	test/test_main.py::test_main_execution	2 ms	
Passed	test/test_ml_surrogate.py::test_train_surrogate_with_mock_data	06 ms	
Passed	test/test_ml_surrogate.py::test_predict_next_state	22 ms	
Passed	test/test_rl_agent.py::test_train_rl_agent	00:00:03	
Passed	test/test_rl_agent.py::test_predict_train_action	1 ms	
Passed	test/test_rl_agent.py::test_train_rl_agent	154 ms	
Passed	test/test_rl_agent.py::test_features_and_predict	31 ms	
Passed	test/test_rl_agent.py::test_observed_callback	2 ms	
Passed	test/test_simulation.py::test_off_bridge_generate_idf	2 ms	
Passed	test/test_simulation.py::test_simulation_engine_get_results	31 ms	
Passed	test/test_simulation_engine.py::test_generate_random_params	1 ms	
Passed	test/test_simulation_engine.py::test_extract_features_targets	5 ms	
Passed	test/test_simulation_engine.py::test_run_simulation	4 ms	
Passed	test/test_simulation_engine.py::test_get_results	10 ms	
Passed	test/test_simulation_engine.py::test_run_batch	0 ms	
Passed	test/test_tabs.py::test_tabs_render	00:00:00	
Passed	test/test_tabs.py::test_tabs_render_clicks	00:00:00	
Passed	test/test_train_agent.py::test_train_agent	5 ms	
Passed	test/test_validator.py::test_validate_simulation_data_good_quality	2 ms	
Passed	test/test_validator.py::test_validate_simulation_data_bad_quality	0 ms	

Рисунок 4.1 – Тестовий звіт

Для оцінки покриття автоматизованого тестування було проведено аналіз за допомогою інструменту Coverage.py. Загальний показник покриття операторів вихідного коду склав 68%, що вважається хорошим показником для науково-місткого програмного забезпечення. Детальний розподіл метрик для кожного файлу можна переглянути на рисунку 4.2.

Coverage report: 68%

Files Functions Classes

coverage.py v7.13.5, created at 2026-05-28 22:48 +0300

File	class	statements	missing	excluded	coverage ▾
ai \ __init__.py	(no class)	0	0	0	100%
ai \ rl_environment.py	(no class)	14	0	0	100%
core \ __init__.py	(no class)	0	0	0	100%
core \ constants.py	(no class)	5	0	0	100%
simulation \ __init__.py	(no class)	0	0	0	100%
simulation \ validator.py	(no class)	15	0	0	100%
ui \ __init__.py	(no class)	0	0	0	100%
ui \ tabs \ __init__.py	(no class)	0	0	0	100%
ui \ tabs \ save_tab.py	(no class)	13	0	0	100%
ui \ designer.py	(no class)	108	6	0	94%
ai \ ml_surrogate.py	(no class)	44	4	0	91%
ai \ rl_environment.py	HVACEnv	113	11	0	90%
ui \ tabs \ geometry_tab.py	(no class)	10	1	0	90%
ai \ rl_agent.py	StreamlitProgressCallback	19	2	0	89%
core \ extract.py	(no class)	53	7	0	87%
simulation \ idf_bridge.py	(no class)	554	91	0	84%
simulation \ simulation_engine.py	(no class)	104	17	0	84%
ai \ ai_engine.py	(no class)	60	12	0	80%
simulation \ simulation_engine.py	SimulationRunner	110	23	0	79%
ai \ rl_agent.py	(no class)	148	37	0	75%
ui \ tabs \ data_gen_tab.py	(no class)	58	18	0	69%
simulation \ validator.py	ModelValidator	160	75	0	53%
ai \ train_agent.py	(no class)	22	14	0	36%
ui \ tabs \ climate_tab.py	(no class)	181	168	0	7%
ui \ tabs \ advisor_tab.py	(no class)	51	48	0	6%
ui \ tabs \ ai_lab_tab.py	(no class)	91	88	0	3%
ai \ rl_agent.py	FixedDeviceWrapper	3	3	0	0%
Total		1936	625	0	68%

Рисунок 4.2 – Звіт покриття модульного тестування

Детальний аналіз звіту демонструє, що найвищий рівень покриття мають модулі обчислювальних та інтелектуальних контурів, де ризик виникнення збоїв логіки чи багів найвищий. Модулі з найнижчим показником покриття – це модулі графічного інтерфейсу. Такі низькі метрики пояснюються декларативною природою рендерингу фреймворку Streamlit, що робить повноцінне модульне тестування технічно складним. Верифікація інтерактивних UI-модулів веб-інтерфейсу вимагає особливих E2E-фреймворків, в той час як в даному проєкті

фокус було зміщено на unit-верифікацію алгоритмічної частини для забезпечення максимальної математичної точності.

Реалізовані 34 тестові сценарії розділено за архітектурними компонентами на 4 умовні напрямки: тестування інтелектуального ІІІ-контур, тестування сурогатного моделювання, тестування пакета координації симуляцій, тестування рівня представлення та менеджменту стану.

Тестування інтелектуального ІІІ-контур: `test_ai_engine.py`, `test_ai_env.py`, `test_rl_agent.py`. У цих модулях верифікується правильність функціонування Gymnasium-середовища HVACEnv та Actor-Critic моделей. Сценарій `test_hvac_env_initialization` перевіряє коректність ініціалізації меж просторів дій та спостережень. Тести `test_reward_calculation_extreme_co2` та `test_reward_calculation_extreme_temperature` контролюють математичну точність нарахування штрафів екологічним контуром середовища при виході параметрів за межі комфорту, тобто перевищення норми у 1000 ppm або падіння температури. Тест `test_normalization_bounds` перевіряє, чи не виходять нормалізовані дані за межі діапазону. Сценарії `test_load_rl_agent`, `test_predict_best_action` та `test_train_rl_agent` перевіряють працездатність Stable Baselines3 та PyTorch контурів: завантаження нейромережі з zip-архіву, генерацію ймовірнісного розподілу дій та бекграунд-навчання політики PPO. Додатково тест `test_generate_xai_explanation` верифікує логіку формування текстових пояснень моделі.

Тестування сурогатного моделювання: `test_ml_surrogate.py`. Сценарій `test_train_surrogate_with_mock_data` імітує процес навчання LightGBM-ансамблю на базі згенерованого синтетичного датасету, а тест `test_predict_next_state` перевіряє точність і швидкість інференсу сурогату.

Тестування пакета координації симуляцій: `test_simulation.py`, `test_simulation_engine.py`, `test_validator.py`. Модуль `test_idf_bridge_generate_idf` контролює синтаксичну правильність генерації текстових блоків EnergyPlus, правильність обходу координат стін Г-подібної геометрії та відсутність кириличних символів. Серія тестів `test_run_simulation`, `test_get_results` та

test_extract_features_targets перевіряє роботу асинхронного виклику підпроцесів ОС, виправлення часових міток 24:00:00 бібліотекою Pandas та конструювання ознак застрягання. Тести test_validate_simulation_data_good_quality та test_validate_simulation_data_bad_quality перевіряють реакцію класу ModelValidator на фізичні аномалії середовища, контролюючи коректність виставлення статусів Pass, Warning та Fail.

Тестування рівня представлення та менеджменту стану: test_designer.py, test_main.py, test_tabs.py. Перевіряють функції ініціалізації стану сесії st.session_state, додавання нових технічних елементів на графічний план Plotly, коректність експорту проекту в JSON та симуляцію кліків по вкладках інтерфейсу для перевірки відсутності критичних помилок рендерингу.

Разом з динамічним модульним тестуванням було проведено статичний аналіз вихідного коду програмного комплексу за допомогою інструменту Radon. Аналіз відбувався за метриками цикломатичної складності коду за методом Маккейба та узагальненого індексу підтримуваності коду. Звіт аналізатора показав відмінну оцінку підтримуваності класу A, та хорошу оцінку складності класу B. Це свідчить про відсутність переускладнених компонентів та надлишкових розгалужень та низьку щільність застережень і помилок. З цього можна зробити висновок, що код є придатним для тривалої підтримки та може легко бути розширеним.

4.2 Сценарій роботи дослідника: підготовка синтетичних даних та навчання моделей машинного навчання

Експлуатаційний сценарій розробника призначений для адміністраторів системи штучного інтелекту програмного комплексу. Він активується через вибір прапорця «Режим розробника» на бічній панелі застосунку, що додає до інтерфейсу дві вкладки, а саме Генерація Даних (Dev) та Навчання Нейромережі (Dev). Робота в цьому режимі починається з генерації синтетичних даних для навчання моделей

під конкретні параметри досліджуваного приміщення. Адміністратор вказує цільову кількість симуляцій, після чого запускає генерацію. Для синтезування різноманітного датасету модуль злегка мутує параметри приміщення, обрає один з п'яти стресових сценаріїв, який випадковим чином додає в маніфест і запускає кілька симуляцій на обчислювальному ядрі EnergyPlus паралельно. Отримані дані трансформуються, очищуються та зберігаються у файл `data/training_dataset.csv`. Після завершення генерації адміністратору показуються метрики оцінки та накопичених даних, матриця кореляцій та випадково обраний шматок сирих даних з масиву, що продемонстровано на рисунку 4.3.

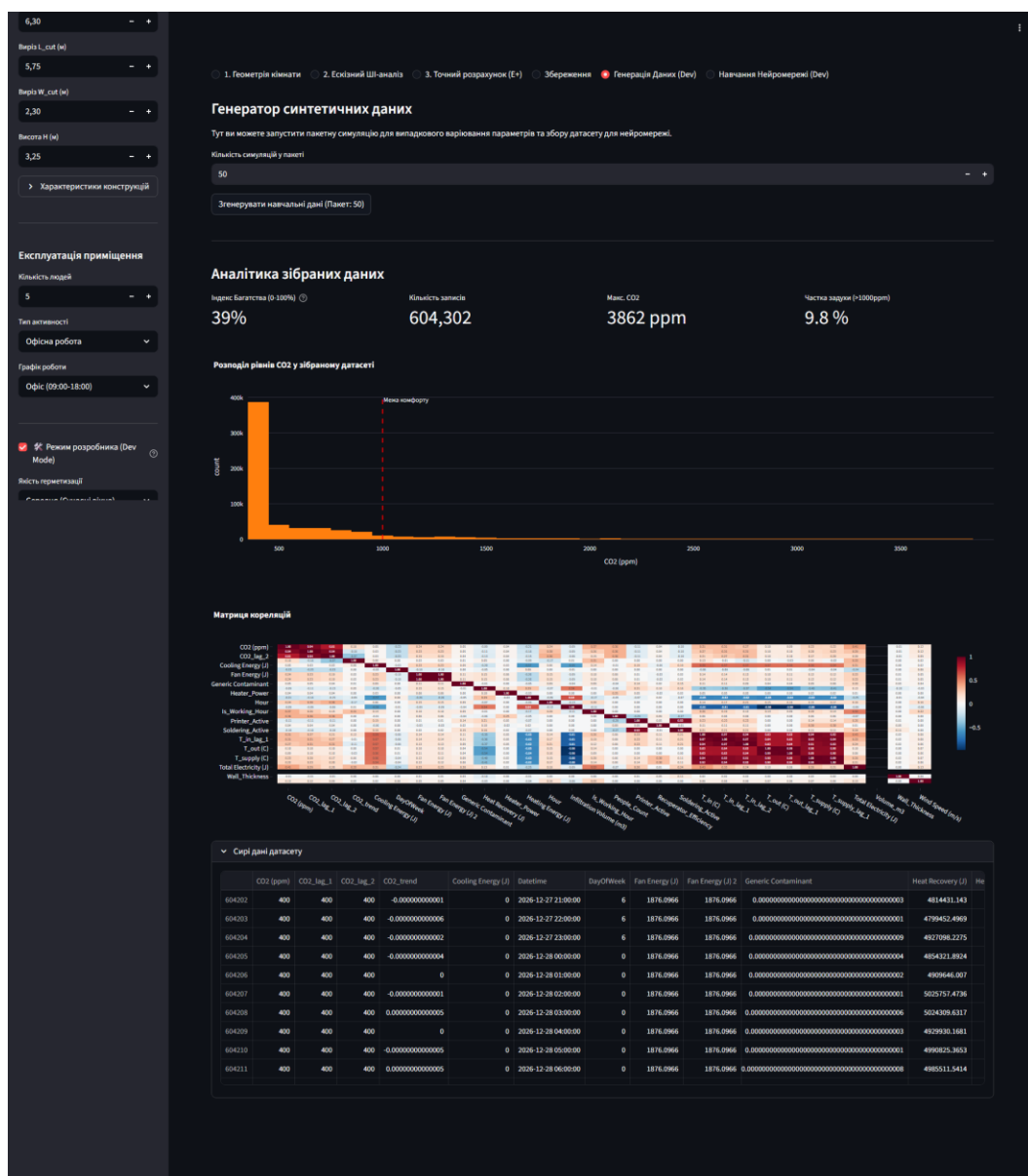


Рисунок 4.3 – Інтерфейс вкладки Генерація Даних

Після формування датасету адміністратор переходить на вкладку Навчання Нейромережі, що умовно розділена на два робочих поля для роботи з сурогатною моделлю, або моделлю навчання з підкріпленням (рис. 4.4).

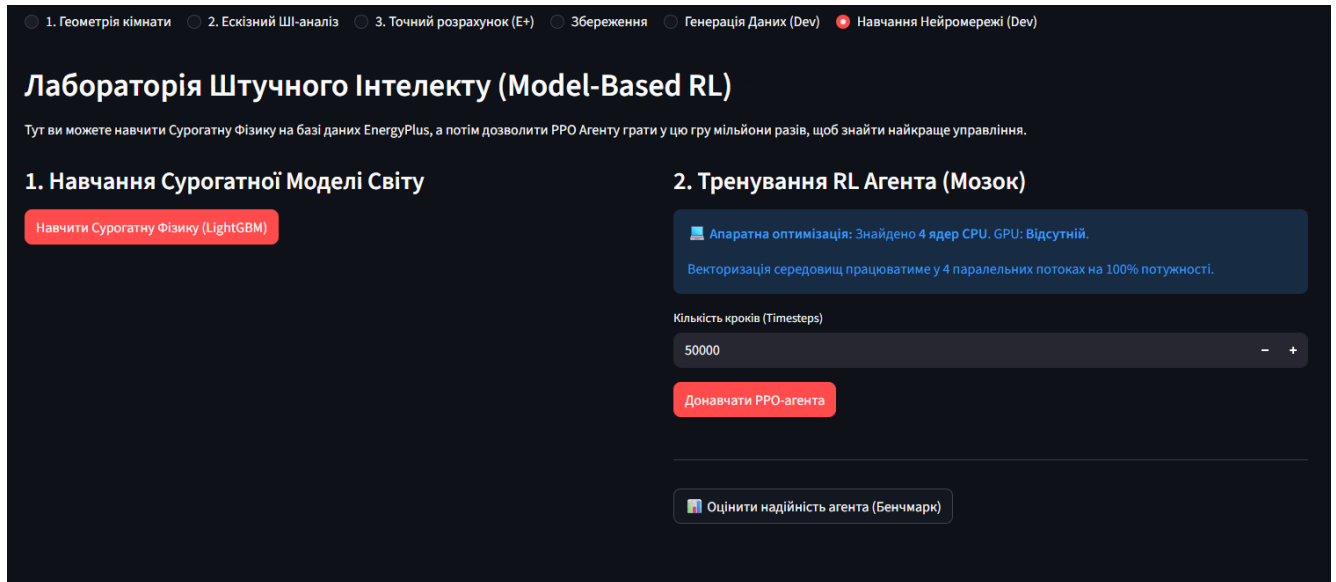


Рисунок 4.4 – Графічний інтерфейс вкладки Навчання Нейромережі

Ліва колонка вкладки відповідає за навчання та тестування сурогатної моделі апроксимації фізичних процесів. При натисканні кнопки Навчити Сурогатну Фізику (LightGBM) викликається функція `ml_surrogate.train_surrogate`. Вона розділяє згенерований масив даних на навчальні та тестувальні вибірки, запускає тренування ансамблю градієнтного бустингу та після успішного навчання виводить на екран метрики оцінки, а саме коефіцієнт детермінації R^2 , похибки MAE та RMSE. Для наочної демонстрації результатів навчання формується діаграма розсіювання, де на вертикальній осі координат відображаються апроксимовані значення, а на горизонтальній осі – реальні значення з тестової вибірки. Результат роботи цього модуля демонструє рисунок 4.5.

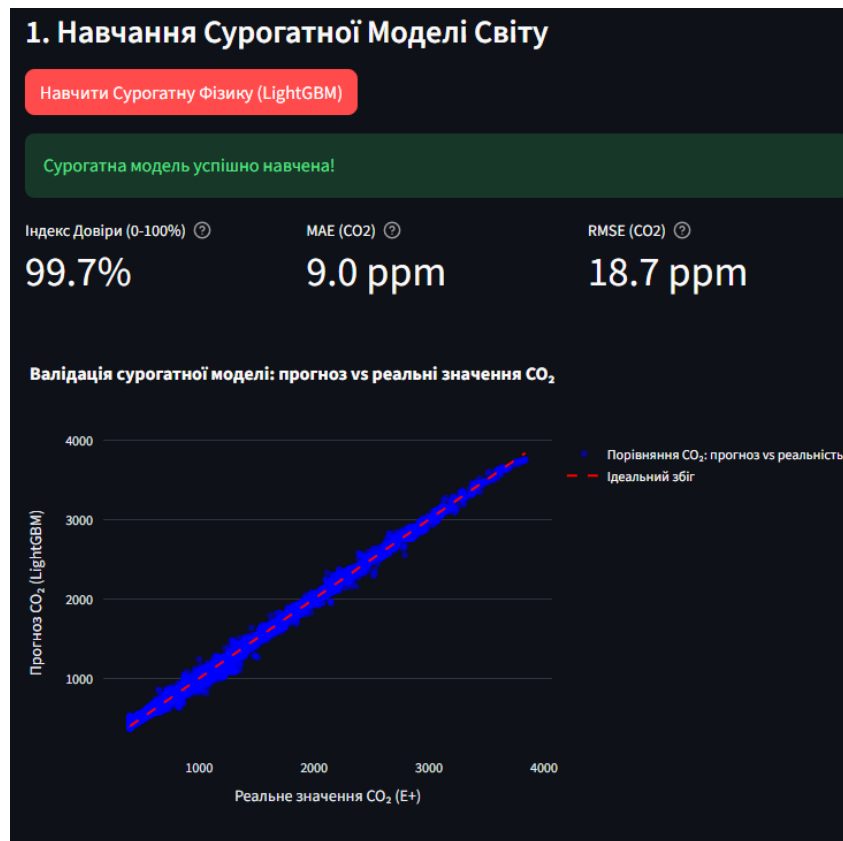


Рисунок 4.5 – Результат навчання сурогатної моделі

Права колонка вкладки відведена під роботу з обчислювальною моделлю за алгоритмом PPO. Система автоматично визначає кількість ядер центрального процесора та активність відеокарти за допомогою засобів multiprocessing та PyTorch для оптимізації багатопотокового навчання та скорочення часу навчання. Запуск алгоритму відбувається в ізольованому фоновому потоці, що дозволяє залишити інтерфейс Streamlit активним при тривалих навчальних сесіях. Для моніторингу навчання в реальному часі було розроблено смугу стану навчання та лінійний графік середньої винагороди за епізод, дані для якого взяті з потокобезпечного буфера shared_state. По закінченню навчання адміністратор може оцінити реалізовані патерни поведінки кнопкою бенчмарку, яка запускає тестування та визначає індекс стійкості агента в екстримальних умовах: стресове навантаження приладів та присутність 30 осіб. Інтерфейс моніторингу продемонстровано на рисунку 4.6.

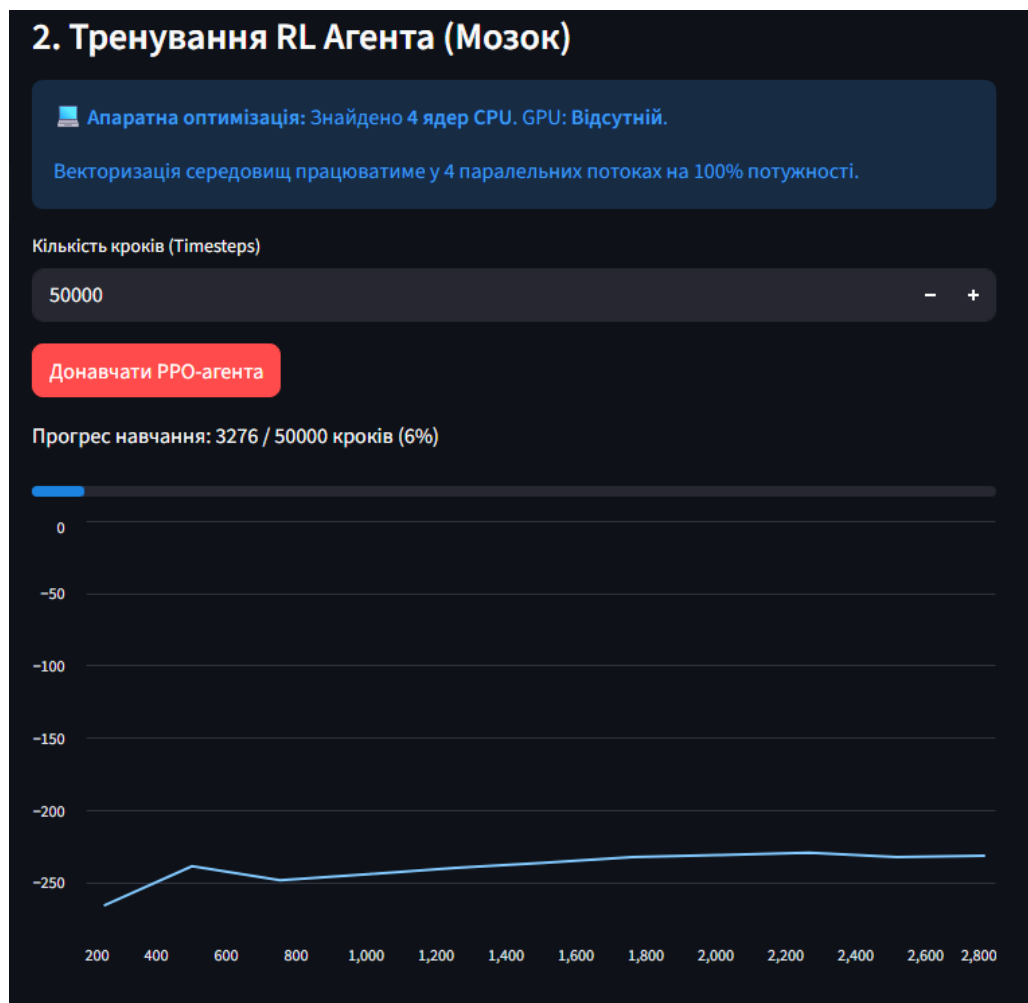


Рисунок 4.6 – Процес навчання PPO-агента

4.3 Сценарій роботи користувача: архітектурне конструювання приміщення та експорт конфігурацій

Експлуатаційний сценарій користувача спрямований на зручне та просте керування геометричними параметрами для задання копії приміщення для цифрового двійника, налаштування поведінки людей в приміщенні, збереження та експорт спроектованих результатів. Логіка цього етапу зосереджена у вкладках Геометрія кімнати та Збереження.

Робочий процес починається з бічної панелі головного модуля застосунку. Вона розділена на секції, для зручності користування та організації елементів керування. У першій секції користувач вводить геометричні параметри кімнати, а

саме довжину, ширину, для Г-подібних кімнат також глибину та ширину вирізу та висоту стелі. У випадючій секції користувач може налаштувати решту параметрів, таких як матеріали стін, їх товщину та чи є вони зовнішніми чи внутрішніми. Інтерфейс цієї частини бокової панелі демонструє рисунок 4.7.

Геометрія та матеріали

Загальна Довжина L (м)
11,75 - +

Загальна Ширину W (м)
6,30 - +

Виріз L_cut (м)
5,75 - +

Виріз W_cut (м)
2,30 - +

Висота H (м)
3,25 - +

▼ Характеристики конструкцій

Матеріал стін
Бетон ▼

λ (Теплопровідність): 1.74 Вт/м·К

ρ (Густина): 2400 кг/м³

c (Теплоємність): 840 Дж/кг·К

Товщина стін (м)
0,38 - +

Рисунок 4.7 – Бічна панель: геометрія та матеріали

Нижче на бічній панелі користувач може налаштувати параметри експлуатації приміщення. Серед них є кількість людей, тип їхньої активності, часові рамки робочого дня, а також ступінь герметизації приміщення та кнопки додавання таких елементів як вікна, рекуператори та джерела тепла, що

викликають функцію `designer.add_element` та автоматично відмальовують доданий елемент. Рисунок 4.8 демонструє цю частину бічної панелі.

Експлуатація приміщення

Кількість людей

5 - +

Тип активності

Офісна робота ▾

Графік роботи

Офіс (09:00-18:00) ▾

☒ Режим розробника (Dev Mode) ?

Якість герметизації

Середня (Сучасні вікна) ▾

Додати об'єкти

Вікно Рекуператор Джерело тепла

Рисунок 4.8 – Бічна панель: експлуатація приміщення

Для точного контролю позиціонування елементів та графічної демонстрації плану приміщення розроблено вкладку Геометрія кімнати. Саме тут графічна компонента Plotly відмальовує планування, кольорове кодування стін та точне розташування усіх доданих об'єктів відповідно до їхнього розміщення за координатами стін. Поточний вигляд графічного відображення лабораторії показує рисунок 4.9.

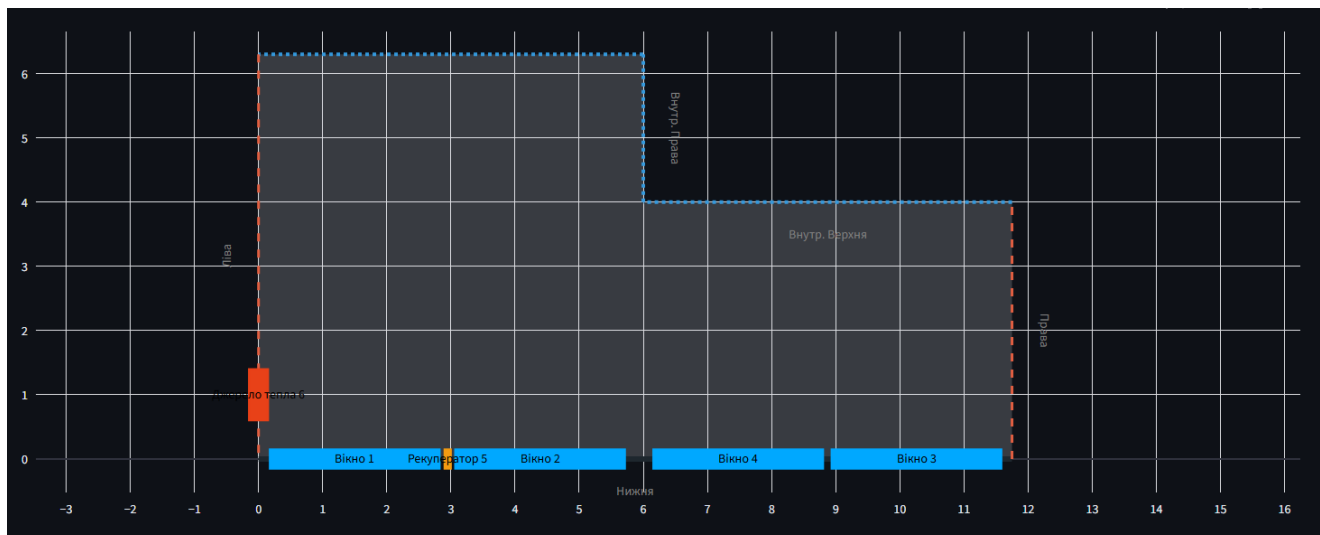


Рисунок 4.9 – Графічне відображення планування приміщення

Під графічним планом розгорнуто інтерактивну таблицю редактора даних `st.data_editor`, яка дозволяє вручну коригувати координати X та Y, лінійну ширину та просторову орієнтацію кожного об'єкта. Будь-яка зміна у таблиці автоматично оновлює обмінну структуру `st.session_state.elements_df` і тригерить миттєве перемальовування Plotly-креслення. Вигляд таблиці елементів представлено на рисунку 4.10.

Таблиця елементів (Швидке редагування)

Тут ви можете вручну змінювати координати, розміри та потужність об'єктів.

Тип	X	Y	Ширина	Орієнтація	Потужність
Вікно	1.5	0	2.65	Горизонтально	1000
Вікно	4.39	0	2.65	Горизонтально	1000
Вікно	10.26	0	2.65	Горизонтально	1000
Вікно	7.48	0	2.65	Горизонтально	1000
Регулятор	2.95	0	0.1	Горизонтально	1000
Джерело тепла	0	1	0.8	Вертикально	1000

Рисунок 4.10 – Таблиця елементів

Після завершення просторового проектування користувач переходить до фінальної стадії формування цифрового двійника у вкладці Збереження, програмна логіка якої зосереджена у модулі `save_tab.py`. Робочий простір надає дві функціональні кнопки: для завантаження стану сесії у стандартизований файл `project_twin.json` через метод `designer.export_project` та для скачування текстового інженерного маніфесту EnergyPlus. Для швидкої інспекції згенерованого коду в

інтерфейсі реалізовано велику текстову область `st.text_area`, яка виводить повний вміст IDF-файлу безпосередньо на екран, що продемонстровано на рисунку 4.11.

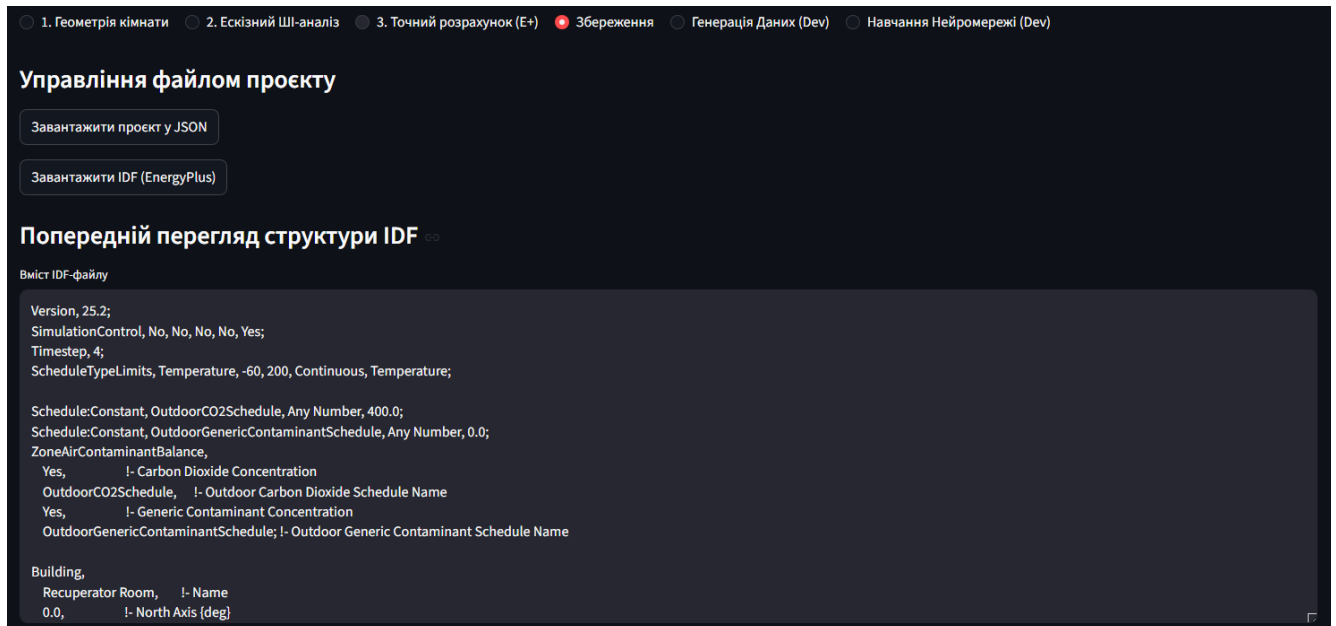


Рисунок 4.11 – Інтерфейс вкладки Збереження

4.4 Проведення автоматизованого підбору обладнання та аналіз радарних діаграм реалістичності мікроклімату

Аналітичний та обчислювальний етап роботи користувача виконується у вкладках Ескізний ШІ-аналіз та Точний розрахунок (E+). Ці компоненти дозволяють провести порівняльне моделювання, реалізувати алгоритм автоматизованого вибору рекуператора засобами машинного навчання та перевірити фізичну достовірність отриманих результатів.

Сценарій ескізного аналізу починається у вкладці Ескізний ШІ-аналіз. Модуль автоматично перевіряє наявність навченої нейромережі РРО за допомогою функції `ai_engine.load_ai_model`. Користувач задає за допомогою слайдера сценарій дня, а саме кількість людей в лабораторії, і натискає кнопку прогнозування. Програма викликає метод `rl_agent.finetune_and_predict`, який аналізує об'єм приміщення і товщину стін для автоматичного підбору оптимальної моделі рекуператора з бази даних. Після цього функція `ai_engine.simulate_24h` розраховує

добові профілі для ручного керування та для інтелектуального автопілота під контролем ШІ. Результати порівняння та візуальне вікно модуля представлено на рисунку 4.12.

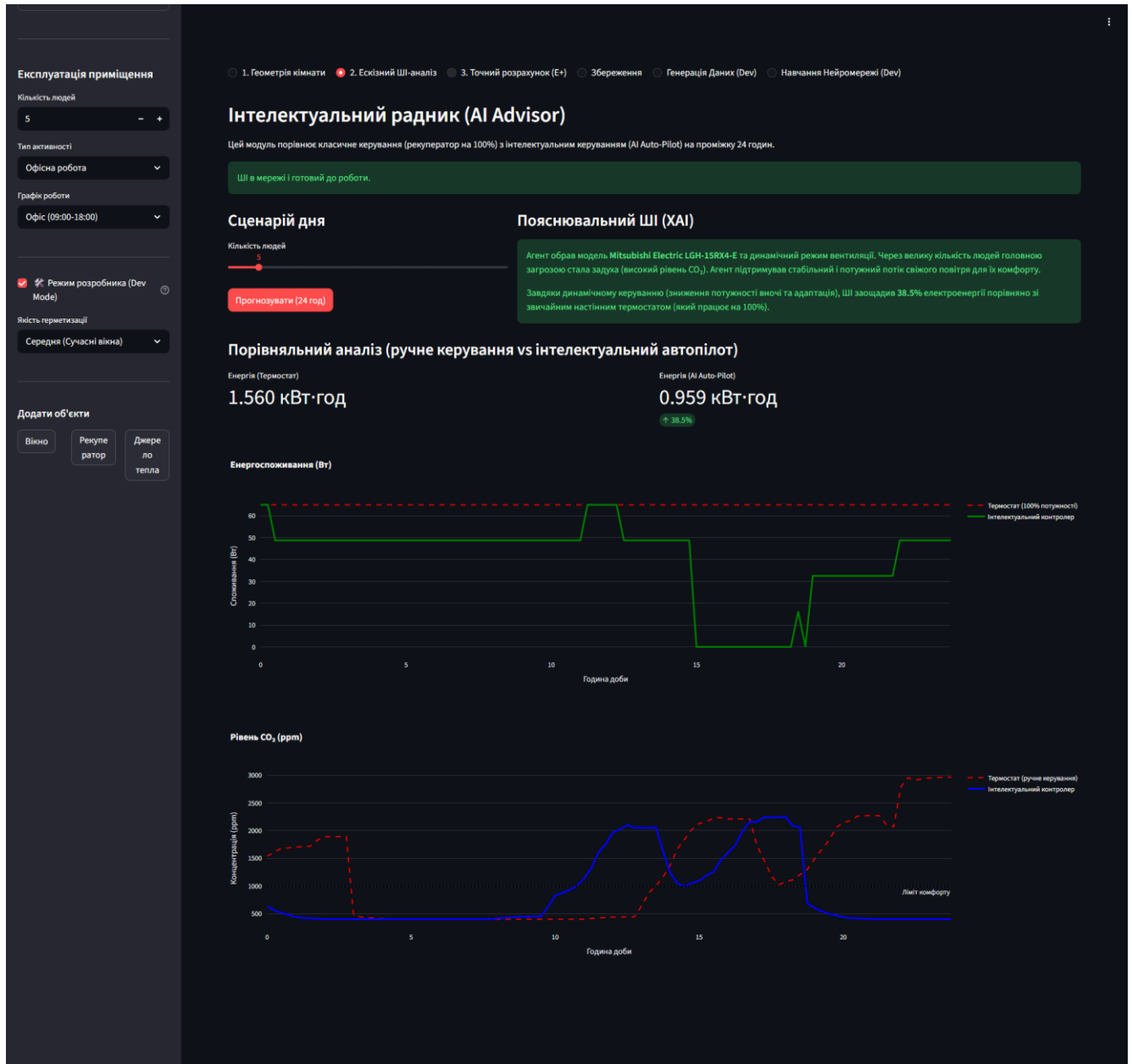


Рисунок 4.12 – Інтерфейс вкладки Ескізний ШІ-аналіз

Для проведення комплексного фізичного моделювання та детального підбору обладнання користувач переходить до вкладки Точний розрахунок (E+). У верхній частині екрана розміщено слайдери температурних уставок термостата та блок інтерактивної математичної моделі, де при ручній зміні потоку повітря та поточної

концентрації CO₂ програма аналітично вираховує годинну зміну параметрів середовища (рис. 4.13).

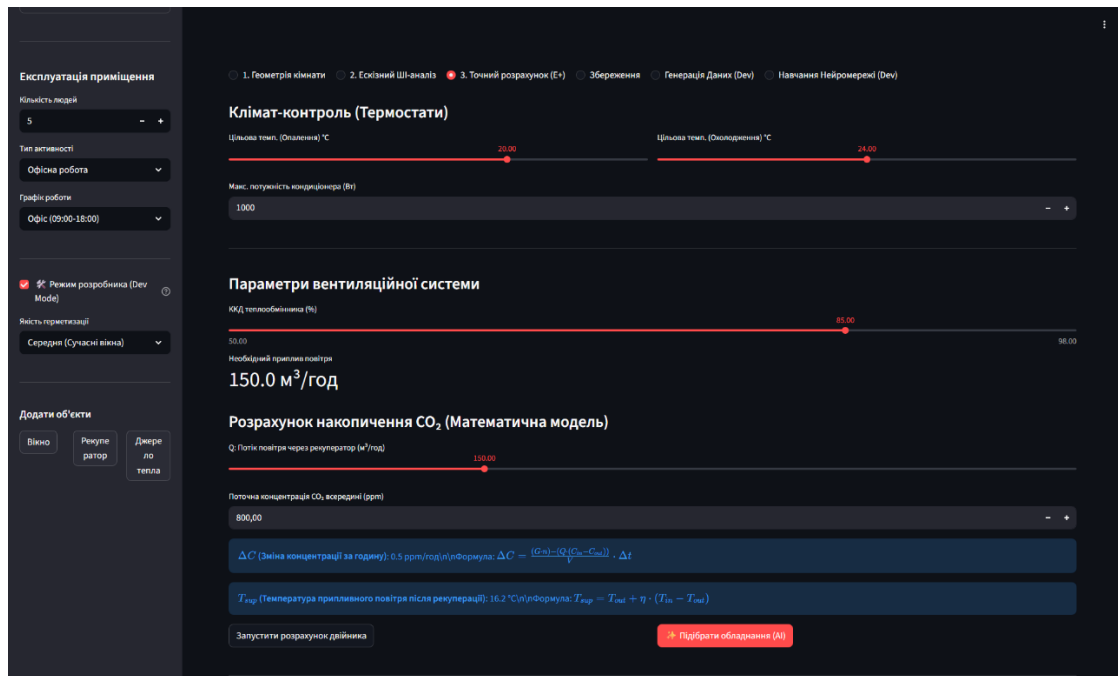


Рисунок 4.13 – Інтерфейс вкладки Точний розрахунок

Натискання кнопки запуску симуляції ініціює створення IDF-структури та запуск фізичного рушія EnergyPlus. Після завершення симуляції всі дані проходять оцінку реалістичності та корисності виконаної роботи завдяки модулю validator.py. Результати верифікації зводяться у радарну полярну діаграму, яку разом із детальним текстовим звітом статусів (Pass, Warning, Fail) представлено на рис. 4.14.

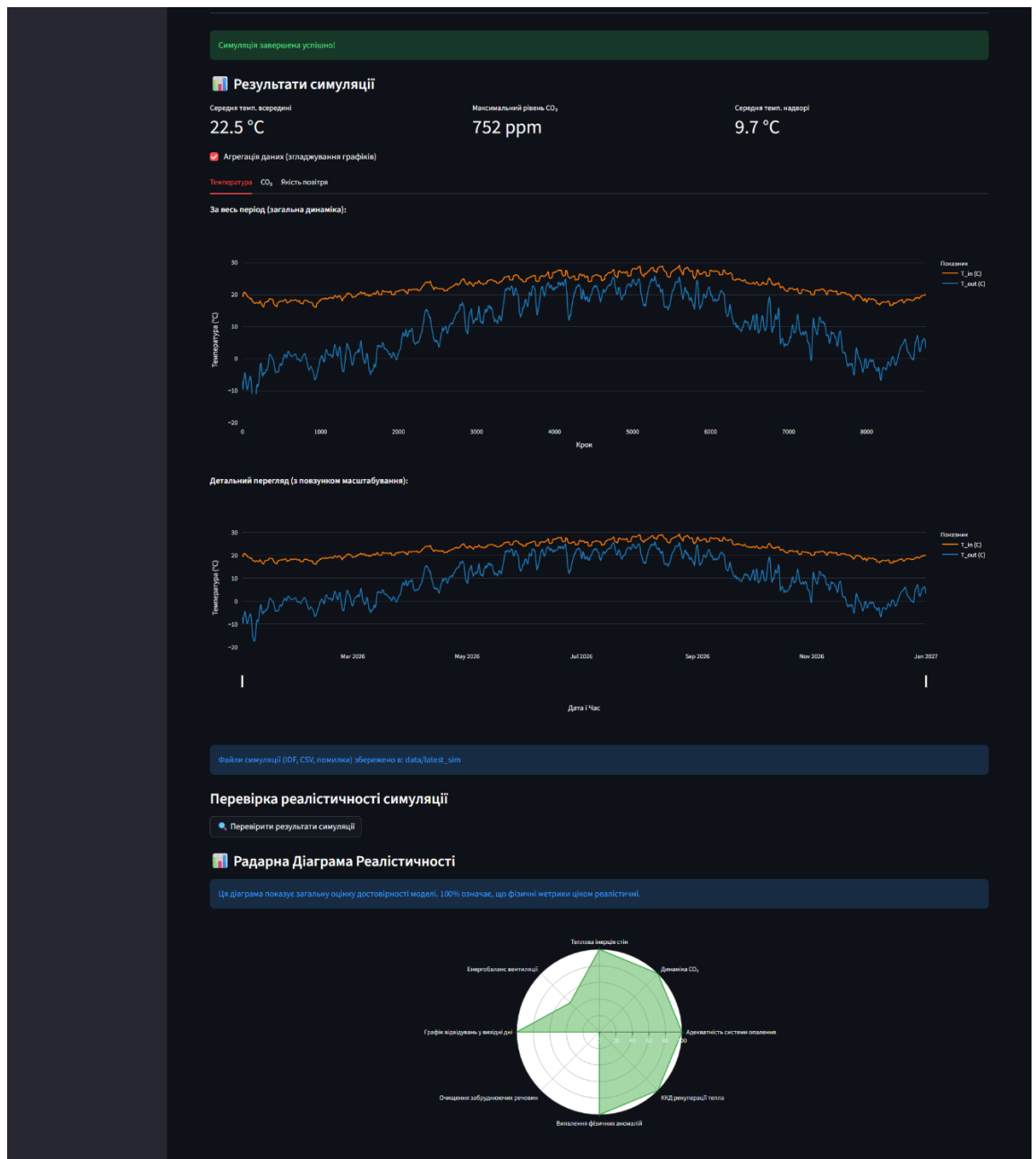


Рисунок 4.14 – Результати симуляції без підбору рекуператора

Натискання кнопки підбору обладнання запускає локальну оптимізацію: ІІІ сканує файл `recuperator_db.json`, визначає найкращий пристрій та автоматично вносить його ККД та максимальний потік у налаштування проекту. Після цього додаток аналогічно звичайній симуляції ініціює створення IDF-структури та запуск фізичного рушія EnergyPlus, але вже з параметрами рекуператора, що набрав максимальну оцінку під час підбору натренованим ІІІ-агентом. Результати такої симуляції наочно продемонстровано на рисунку 4.15.

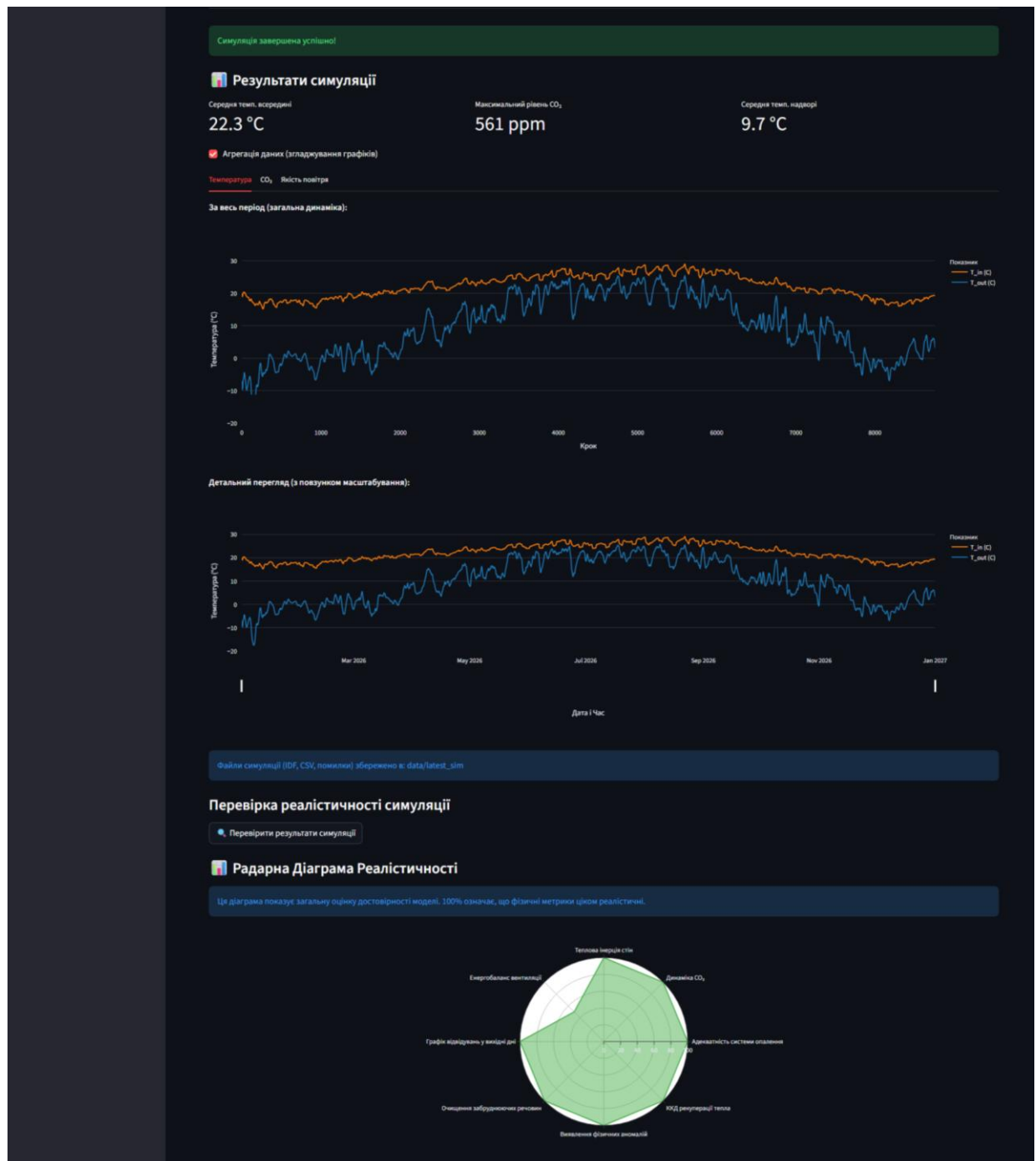


Рисунок 4.15 – Результати симуляції після підбору рекуператора

Аналіз результатів показав значні позитивні зміни в порівненні з симуляцією без підбору рекуператора. Це свідчить про те, що алгоритм підбору обладнання не просто працює, але й робить це ефективно та точно.

Висновки до розділу 4

У четвертому розділі проведено комплексну верифікацію та експериментальне тестування працездатності програмного комплексу HIVanalyzer. Результати автоматизованого тестування за допомогою фреймворку PyTest підтвердили повну коректність обчислювальної логіки у 34 розроблених unit-сценаріях із загальним покриттям операторів на рівні 68%, а статичний аналіз кодової бази утилітою Radon зафіксував найвищий клас підтримуваності коду А. Успішна перевірка інтерфейсних сценаріїв продемонструвала відсутність збоїв при взаємодії користувача з Plotly-конструктором геометрії, модулями збереження даних та фоновим контуром навчання нейромереж РРО.

Порівняльний аналіз імітаційного моделювання показує, що автоматизований підбір рекуператора методами машинного навчання дозволив суттєво покращити якість повітря в лабораторії, знизивши пікову концентрацію CO₂ з 752 ppm до 561 ppm. Оптимізація режимів вентиляції відбулася без переохолодження приміщення, забезпечивши утримання стабільної внутрішньої температури в межах 22.3-22.5 градусів Цельсія. Робота інтелектуального автопілота супроводжувалася досягненням максимальної оцінки 100% за полярною радарною діаграмою вбудованого модуля фізичної валідації, що повністю підтверджує математичну достовірність, високу інженерну ефективність та доцільність практичного використання розробленого продукту.

ВИСНОВКИ

У дипломній роботі успішно вирішено актуальну науково-технічну задачу автоматизації та оптимізації процесу вибору системи вентиляції з рекуперацією тепла приміщення для інженерної лабораторії SmartEnergyLab. Шляхом поєднання технології цифрових двійників та алгоритмів машинного навчання розроблено та верифіковано програмний комплекс HlV analyzer, який дозволяє здійснювати точний підбору припливно-витяжних систем вентиляції з рекуперацією тепла. Дослідження термодинамічних процесів у закритих просторах підтвердило нелінійний характер накопичення антропогенних та технологічних забруднювачів, що зумовило вибір комплексу EnergyPlus як базового математичного ядра для врахування теплової інерції стін, просочення повітря через негерметичність огорожувальних конструкцій та ефектів масообміну в приміщеннях складної геометрії.

Впровадження модульної архітектури на основі патернів Фасад і Адаптер дозволило повністю ізолювати низькорівневі обчислювальні процеси симулятора від високорівневої логіки штучного інтелекту та реактивного інтерфейсу Streamlit. В межах роботи виконано формалізацію параметрів інженерної лабораторії. Створена гнучка підсистема експлуатаційних сценаріїв через EMS-макриси та компактні розклади дозволила реалістично відтворити профілі відвідуваності та випадкові технологічні сплески виділення забруднюючих речовин.

Для побудови інтелектуального контуру було інтегровано спеціалізоване навчальне середовище HVACEnv на базі стандарту Gymnasium. З метою подолання низької швидкості прямих термодинамічних розрахунків реалізовано сурогатну модель світу на базі ансамблевого алгоритму LightGBM, що дозволило значно скоротити час обчислення одного кроку при збереженні високої точності апроксимації. Керуючий III-контур на основі алгоритму Proximal Policy Optimization реалізує проактивну стратегію управління, гнучко балансує між

санітарними обмеженнями та критеріями енергозбереження під час роботи рекуператора.

Засобами фреймворку Streamlit створено інтерактивний графічний інтерфейс програми з централізованим сховищем стану сесії, що забезпечує стабільний рух інформаційних потоків від двовимірного конструктора геометрії до аналітичних модулів. Процес донавчання агента штучного інтелекту успішно ізольовано у фоновому потоці операційної системи із передачею метрик через потокобезпечний буфер `shared_state`, що ліквідувало проблему завмирання веб-інтерфейсу та усунуло помилки розриву з'єднання з браузером під час тривалих обчислювальних сесій.

Динамічне тестування комплексу за допомогою фреймворку PyTest підтвердило обчислювальну стійкість системи, продемонструвавши успішне виконання всіх 34 unit-сценаріїв із загальним покриттям операторів кодової бази на рівні 68%. Комплексний порівняльний аналіз моделювання показав, що автоматизований підбір оптимальної моделі рекуператора з бази даних у поєднанні з проактивним ШІ-автопілотом дозволив знизити пікову концентрацію вуглекислого газу з 752 ppm до 561 ppm при збереженні стабільної внутрішньої температури в межах від 22.3°C до 22.5°C. При цьому вбудований модуль валідації зафіксував розширення площі покриття радарної діаграми реалістичності, що підтверджує повну ліквідацію фізичних аномалій та високу практичну цінність розробленого застосунку для реальних кіберфізичних систем управління мікрокліматом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASHRAE Standard 62.1-2022: Ventilation for Acceptable Indoor Air Quality // American Society of Heating, Refrigerating and Air-Conditioning Engineers. - 2022. - 94 p. URL: https://www.ashrae.org/file%20library/technical%20resources/standards%20and%20guidelines/standards%20addenda/62_1_2022_w_20250930.pdf
2. Incropera F. P., Dewitt D. P. Fundamentals of Heat and Mass Transfer // John Wiley & Sons. - 2011. - 1048 p. URL: <https://www.wiley.com/en-us/shop/general-introductory-mechanical-engineering/fundamentals-of-heat-and-mass-transfer-8th-edition-p-9781119353881>
3. Grieves M. Digital Twin: Mitigating Unpredictable Unforeseen Tactics in the Product Life Cycle // White Paper. - 2014. - P. 1-19. URL: https://www.researchgate.net/publication/306223791_Digital_Twin_Mitigating_Unpredictable_Undesirable_Emergent_Behavior_in_Complex_Systems
4. Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction // MIT Press. - 2018. - 552 p. URL: <https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf>
5. Crawley D. B., Lawrie L. K., Pedersen C. O., Winkelmann F. C. EnergyPlus: creating a new-generation building energy simulation program // Energy and Buildings. - 2001. - Vol. 33, no. 4. - P. 319-331. URL: <https://www.sciencedirect.com/science/article/abs/pii/S0378778800001146>
6. Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. Proximal Policy Optimization Algorithms // arXiv preprint arXiv:1707.06347. - 2017. - P. 1-12. URL: <https://arxiv.org/abs/1707.06347>
7. Streamlit API Reference Documentation // Streamlit Inc. - 2026. URL: <https://docs.streamlit.io>
8. Gymnasium Documentation: A standard interface for reinforcement learning environments // Farama Foundation. - 2026. URL: <https://gymnasium.farama.org>

9. pandas: powerful Python data analysis toolkit // pandas development team. - 2026. URL: <https://pandas.pydata.org/docs/>
10. LightGBM Documentation: Light Gradient Boosting Machine // Microsoft Corporation. - 2026. URL: <https://lightgbm.readthedocs.io>
11. Plotly Python Graphing Library Documentation // Plotly Technologies Inc. - 2025. URL: <https://plotly.com/python/>
12. pytest: helps you write better programs // PyTest Developer Community. - 2026. URL: <https://docs.pytest.org/>
13. NumPy Reference Guide // NumPy Developers. - 2026. URL: <https://numpy.org/doc/>
14. Stable Baselines3 Documentation // Stable Baselines3 Contributors. - 2026. URL: <https://stable-baselines3.readthedocs.io>
15. PyTorch Documentation // PyTorch Foundation. - 2026. URL: <https://pytorch.org/docs>
16. Radon Documentation: Code Metrics in Python // Radon Developer Community. - 2025. URL: <https://radon.readthedocs.io/en/latest/>

ДОДАТОК А

Програмний додаток для підбору рекуператора методами машинного навчання на основі цифрового двійника приміщення

Програмний код

Аркушів 14

Київ – 2026

Main.py

```
"""
Головний модуль застосунку HIVanalyzer (Streamlit).
Забезпечує користувацький інтерфейс для конфігурації приміщення, візуалізації,
запуску симуляцій EnergyPlus та взаємодії зі штучним інтелектом.
"""

import streamlit as st

from core import constants
from ui import designer
from ui.tabs import (
    advisor_tab,
    ai_lab_tab,
    climate_tab,
    data_gen_tab,
    geometry_tab,
    save_tab,
)

st.set_page_config(page_title="Recuperator Optimizer", layout="wide")
designer.init_session_state()

# --- SIDEBAR: ВВІД ПАРАМЕТРІВ ---
st.sidebar.header("Конфігурація системи")
with st.sidebar.expander("Шляхи EnergyPlus"):
    st.session_state.project_settings["eplus_exe"] = st.text_input(
        "Шлях до energyplus.exe",
        st.session_state.project_settings.get(
            "eplus_exe", constants.SIMULATION["eplus_exe"]
        ),
    )
    st.session_state.project_settings["weather_file"] = st.text_input(
        "Шлях до файлу погоди",
        st.session_state.project_settings.get(
            "weather_file", constants.SIMULATION["weather_file"]
        ),
    )

st.sidebar.markdown("---")
if st.sidebar.button("Завантажити налаштування з data/default.json"):
    designer.load_default_project("data/default.json")
    st.session_state["default_loaded"] = True

st.sidebar.header("Геометрія та матеріали")
room_l = st.sidebar.number_input(
    "Загальна Довжина L (м)",
    2.0,
    30.0,
    st.session_state.project_settings.get("room_l", 8.0),
    0.1,
    key="room_l",
)
room_w = st.sidebar.number_input(
    "Загальна Ширина W (м)",
    2.0,
    30.0,
    st.session_state.project_settings.get("room_w", 6.0),
    0.1,
    key="room_w",
)
```

```

room_l_cut = st.sidebar.number_input(
    "Виріз L_cut (м)",
    0.0,
    20.0,
    st.session_state.project_settings.get("room_l_cut", 3.0),
    0.1,
    key="room_l_cut",
)
room_w_cut = st.sidebar.number_input(
    "Виріз W_cut (м)",
    0.0,
    20.0,
    st.session_state.project_settings.get("room_w_cut", 3.0),
    0.1,
    key="room_w_cut",
)
room_h = st.sidebar.number_input(
    "Висота H (м)",
    2.0,
    5.0,
    st.session_state.project_settings.get("room_h", 2.8),
    0.1,
    key="room_h",
)

st.session_state.project_settings["room_l"] = room_l
st.session_state.project_settings["room_w"] = room_w
st.session_state.project_settings["room_l_cut"] = room_l_cut
st.session_state.project_settings["room_w_cut"] = room_w_cut
st.session_state.project_settings["room_h"] = room_h

with st.sidebar.expander("Характеристики конструкцій"):
    mat_options = list(constants.MATERIALS.keys())
    st.session_state.project_settings["wall_material"] = st.selectbox(
        "Матеріал стін",
        mat_options,
        index=mat_options.index(
            st.session_state.project_settings.get("wall_material", mat_options[0])
        ),
    )

    # Відображення характеристик обраного матеріалу
    sel_mat_props = constants.MATERIALS[
        st.session_state.project_settings["wall_material"]
    ]
    st.info(
        f"λ (Теплопровідність): {sel_mat_props['conductivity']} Вт/м·К\n\n"
        f"ρ (Густина): {sel_mat_props['density']} кг/м³\n\n"
        f"с (Теплоємність): {sel_mat_props['specific_heat']} Дж/кг·К"
    )

    st.session_state.project_settings["wall_thickness"] = st.number_input(
        "Товщина стін (м)", 0.1, 1.0, 0.38
    )

    st.markdown("Налаштування стін (Г-форма: 6 стін):")
    wall_labels = ["Нижня", "Права", "Внутр. Верхня", "Внутр. Права", "Верхня",
"Ліва"]
    for i, label in enumerate(wall_labels):
        c1, c2 = st.columns([1, 1])
        st.session_state.project_settings[f"wall_type_{i}"] = c1.selectbox(

```

```

        f"{label}",
        ["Зовнішня", "Внутрішня"],
        index=0
        if st.session_state.project_settings.get(f"wall_type_{i}", "Зовнішня")
        == "Зовнішня"
        else 1,
        key=f"w_type_{i}",
    )
    st.session_state.project_settings[f"wall_mat_{i}"] = c2.selectbox(
        "Мат.",
        ["Базовий", "Скло"],
        index=0
        if st.session_state.project_settings.get(f"wall_mat_{i}", "Базовий")
        == "Базовий"
        else 1,
        key=f"w_mat_{i}",
    )

st.sidebar.markdown("---")
st.sidebar.header("Експлуатація приміщення")
room_occupants_default = st.session_state.project_settings.get("occupants", 10)
room_activity_default = st.session_state.project_settings.get(
    "activity_level", "Офісна робота"
)

occupants = st.sidebar.number_input(
    "Кількість людей", 0, 100, room_occupants_default, key="occupants"
)

activity_options = ["Відпочинок", "Офісна робота", "Легка праця", "Важка праця"]
activity_index = (
    activity_options.index(room_activity_default)
    if room_activity_default in activity_options
    else 1
)

activity_level = st.sidebar.selectbox(
    "Тип активності", activity_options, index=activity_index, key="activity_level"
)

st.session_state.project_settings["occupants"] = occupants
st.session_state.project_settings["activity_level"] = activity_level

schedule_options = ["Офіс (09:00-18:00)", "Житлове (24/7)", "Серверна/Склад (24/7)"]
schedule_default = st.session_state.project_settings.get(
    "schedule_type", "Офіс (09:00-18:00)"
)

sched_index = (
    schedule_options.index(schedule_default)
    if schedule_default in schedule_options
    else 0
)

schedule_type = st.sidebar.selectbox(
    "Графік роботи", schedule_options, index=sched_index, key="schedule_type"
)

st.session_state.project_settings["schedule_type"] = schedule_type

st.sidebar.markdown("---")
dev_mode = st.sidebar.checkbox(
    "🔧 Режим розробника (Dev Mode)",
    value=False,
    help="Відображає вкладки для тренування нейромереж та генерації датасетів.",
)

```

```

)

infiltr_options = [
    "Низька (Старий будинок з протягами)",
    "Середня (Сучасні вікна)",
    "Висока (Пасивний будинок)",
]
infiltr_default = st.session_state.project_settings.get(
    "infiltration", "Середня (Сучасні вікна)"
)
infiltr_idx = (
    infiltr_options.index(infiltr_default) if infiltr_default in infiltr_options else
    1
)
st.session_state.project_settings["infiltration"] = st.sidebar.selectbox(
    "Якість герметизації", infiltr_options, index=infiltr_idx,
    key="infiltration_level"
)

# --- УПРАВЛІННЯ ЕЛЕМЕНТАМИ ---
st.sidebar.markdown("---")
st.sidebar.subheader("Додати об'єкти")
col_add1, col_add2, col_add3 = st.sidebar.columns(3)
if col_add1.button("Вікно"):
    designer.add_element("Вікно", room_l, room_w)
if col_add2.button("Рекуператор"):
    designer.add_element("Рекуператор", room_l, room_w)
if col_add3.button("Джерело тепла"):
    designer.add_element("Джерело тепла", room_l, room_w)

# --- ОСНОВНИЙ РОБОЧИЙ ПРОСТІР ---
if dev_mode:
    tab_options = [
        "1. Геометрія кімнати",
        "2. Ескізний III-аналіз",
        "3. Точний розрахунок (E+)",
        "Збереження",
        "Генерація Даних (Dev)",
        "Навчання Нейромережі (Dev)",
    ]
else:
    tab_options = [
        "1. Геометрія кімнати",
        "2. Ескізний III-аналіз",
        "3. Точний розрахунок (E+)",
        "Збереження",
    ]

selected_tab = st.radio("Навігація", tab_options, horizontal=True,
    label_visibility="collapsed")

if selected_tab == "1. Геометрія кімнати":
    geometry_tab.render(room_l, room_w, room_l_cut, room_w_cut)

elif selected_tab == "2. Ескізний III-аналіз":
    advisor_tab.render(room_l, room_w, room_h, room_l_cut, room_w_cut)

elif selected_tab == "3. Точний розрахунок (E+)":
    climate_tab.render(room_l, room_w, room_h, room_l_cut, room_w_cut)

elif selected_tab == "Збереження":

```

```

        save_tab.render(room_l, room_w, room_h, room_l_cut, room_w_cut)

elif selected_tab == "Генерація Даних (Dev)":
    data_gen_tab.render(room_l, room_w, room_h, room_l_cut, room_w_cut)

elif selected_tab == "Навчання Нейромережі (Dev)":
    ai_lab_tab.render(room_l, room_w, room_h, room_l_cut, room_w_cut)

```

Ml_surrogate.py

```

import os

import joblib
import numpy as np
import pandas as pd
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.multioutput import MultiOutputRegressor

SURROGATE_MODEL_PATH = "models/surrogate_physics.joblib"

# Вхідні ознаки: поточний стан середовища + параметри дії
FEATURES = [
    # Ознаки стану середовища
    "T_out (C)",
    "T_in_lag_1",          # Температура всередині 1 годину тому
    "T_in_lag_2",          # Температура всередині 2 години тому – теплова інерція
    # Температура припливу після рекуперації – прямий сигнал
    "T_supply_lag_1",
    "CO2_lag_1",           # Концентрація CO2 1 годину тому
    "CO2_lag_2",           # Концентрація CO2 2 години тому – тренд
    # Ознаки дії (параметри обраного обладнання)
    "Recuperator_Efficiency",
    "People_Count",
    "Hour",
    "DayOfWeek",          # День тижня: вихідні мають принципово інший CO2-профіль
    "Is_Working_Hour",
    "Volume_m3",
    "Wall_Thickness",
    "Soldering_Active",
    "Printer_Active",
    "Heater_Power",
]

# Цільові змінні (прогноз на наступний часовий крок)
TARGETS = ["T_in (C)", "CO2 (ppm)"]

def train_surrogate(dataset_path: str = "data/training_dataset.csv"):
    """
    Навчає Суррогатну Фізику (Random Forest) на базі CSV з EnergyPlus.
    Ця модель буде слугувати "Швидким Симулятором" (Світом) для RL Агента.
    """
    if not os.path.exists(dataset_path):
        return False, "Файл датасету не знайдено. Спочатку згенеруйте дані."

    df = pd.read_csv(dataset_path)

```

```

# Перевірка наявності всіх вхідних ознак у датасеті
for f in FEATURES:
    if f not in df.columns:
        # Коректна обробка відсутньої ознаки з інформативним повідомленням
        return False, f"Відсутня колонка у масиві даних: {f}"

if "CO2 (ppm)" not in df.columns:
    return False, "Відсутня цільова колонка CO2 (ppm)"

# Для спрощення, будемо навчати просто передбачати абсолютне значення CO2 на
наступному кроці,
# хоча dataset logging збирає CO2 поточного кроку.
# Так як це Time-Series, y = df['T_in (C)'], df['CO2 (ppm)']
# а X = df['T_in_lag_1'], df['CO2_lag_1']

# Базова фільтрація вхідних ознак
X = df[FEATURES]
# Цільові змінні: температура та рівень CO2 на наступному часовому кроці
y = df[["T_in (C)", "CO2 (ppm)"]]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1, random_state=42
)

base_model = LGBMRegressor(
    # 300 дерев забезпечують достатню виразну здатність для 16 фічей,
    # max_depth=6 обмежує заглиблення і знижує overfitting порівняно з
max_depth=10,
    n_estimators=300,
    max_depth=6,
    learning_rate=0.05,
    subsample=0.8,          # стохастична вибірка підвибірки – регуляризація
    colsample_bytree=0.8,   # випадкове використання фічей по дереву
    verbose=-1,
    random_state=42
)
model = MultiOutputRegressor(base_model)
model.fit(X_train, y_train)

score = model.score(X_test, y_test)

os.makedirs(os.path.dirname(SURROGATE_MODEL_PATH), exist_ok=True)
joblib.dump(model, SURROGATE_MODEL_PATH)

# Передбачення на тест-вибірці для побудови графіків валідації
y_pred = model.predict(X_test)

mae_t = mean_absolute_error(y_test["T_in (C)"], y_pred[:, 0])
rmse_t = np.sqrt(mean_squared_error(y_test["T_in (C)"], y_pred[:, 0]))

mae_co2 = mean_absolute_error(y_test["CO2 (ppm)"], y_pred[:, 1])
rmse_co2 = np.sqrt(mean_squared_error(y_test["CO2 (ppm)"], y_pred[:, 1]))

metrics = {
    "surrogate_precision_r2": score,
    "dataset_size": len(df),
    "mae_t": mae_t,
    "rmse_t": rmse_t,
    "mae_co2": mae_co2,
    "rmse_co2": rmse_co2,
    "y_test_t_in": y_test["T_in (C)"].values.tolist(),

```

```

        "y_pred_t_in": y_pred[:, 0].tolist(),
        "y_test_co2": y_test["CO2 (ppm)"].values.tolist(),
        "y_pred_co2": y_pred[:, 1].tolist(),
    }

    return True, metrics

def load_surrogate():
    """Завантажує сурогатну модель-Світ з диска."""
    if os.path.exists(SURROGATE_MODEL_PATH):
        return joblib.load(SURROGATE_MODEL_PATH)
    return None

def predict_next_state(model, state_dict: dict):
    """
    Дає швидкий прогноз (1 мс) наступного стану замість Е+.
    """
    input_df = pd.DataFrame([state_dict])[FEATURES]
    preds = model.predict(input_df)[0]

    return {"T_in_next": preds[0], "CO2_next": preds[1]}

```

train_agent.py

```

import os

from stable_baselines3 import PPO
from stable_baselines3.common.vec_env import SubprocVecEnv

from ai.rl_environment import HVACEnv

def make_env():
    def _init():
        return HVACEnv()

    return _init

if __name__ == "__main__":
    print("Starting PPO Training Pipeline...")

    # Крок 3: Hardware Optimization (Многопоточність)
    num_cpu = 4
    print(f"Initializing {num_cpu} parallel environments...")
    env = SubprocVecEnv([make_env() for i in range(num_cpu)])

    # Крок 2: Конфігурація RL Агента (PPO)
    policy_kwargs = dict(net_arch=dict(pi=[64, 64], vf=[64, 64]))

    print("Configuring PPO Agent with tuned hyperparameters...")
    model = PPO(
        "MlpPolicy",
        env,
        verbose=1,
        tensorboard_log="./logs/",
        learning_rate=3e-4,
        n_steps=2048,
        batch_size=64,
        policy_kwargs=policy_kwargs,
    )

```

```

        device="cpu"
    )

    print("Training Agent... Monitor progress using: tensorboard --logdir
./logs/")
    # Почни з 100 тис. для тесту
    model.learn(total_timesteps=100000)

    # Збереження моделі
    os.makedirs("models", exist_ok=True)
    save_path = "models/recuperator_agent_v1"
    model.save(save_path)
    print(f"Training completed! Model successfully saved to {save_path}.zip")

```

simulation_engine.py

```

"""
Модуль керування симуляціями EnergyPlus.
Відповідає за запуск процесів симуляції у фоновому режимі, збір результатів
та їх парсинг у формат pandas DataFrame. Включає інструменти пакетної генерації.
"""
import os
import random
import subprocess

import pandas as pd

from core import constants

def run_simulation(
    idf_content: str, eplus_exe: str = None, epw_path: str = None,
    sim_dir_override: str = None
) -> (bool, str, str):
    """
    Runs EnergyPlus simulation.
    Returns: (success_bool, csv_data_or_error_message, sim_dir_path)
    """
    if sim_dir_override:
        sim_dir = sim_dir_override
    else:
        sim_dir = os.path.join("data", "latest_sim")

    if not os.path.exists(sim_dir):
        os.makedirs(sim_dir)

    idf_path = os.path.join(sim_dir, "in.idf")

    with open(idf_path, "w", encoding="utf-8") as f:
        f.write(idf_content)

    exe_path = eplus_exe if eplus_exe else constants.SIMULATION["eplus_exe"]
    weather = epw_path if epw_path else constants.SIMULATION["weather_file"]

    if not os.path.exists(exe_path):
        return False, f"Вказаний шлях до EnergyPlus не знайдено:\n{exe_path}",
    sim_dir

    cmd = [
        exe_path,
        "-x",
        "-r",

```

```

        "-d",
        sim_dir,
    ]
    if os.path.exists(weather):
        cmd.extend(["-w", weather])
    cmd.append(idf_path)

    try:
        result = subprocess.run(cmd, capture_output=True, text=True, check=False)

        if result.returncode != 0:
            err_path = os.path.join(sim_dir, "eplusout.err")
            err_msg = f"EnergyPlus завершився з кодом {result.returncode}.\n"
            if os.path.exists(err_path):
                with open(err_path, "r", encoding="utf-8") as ef:
                    lines = ef.readlines()
                    err_msg += "".join(lines[:10]) # Get first 10 lines of error
            else:
                err_msg += result.stderr
            return False, err_msg, sim_dir

        csv_path = os.path.join(sim_dir, "eplusout.csv")
        if os.path.exists(csv_path):
            return True, csv_path, sim_dir
        else:
            return (
                False,
                "Симуляція пройшла, але файл eplusout.csv не знайдено.",
                sim_dir,
            )

    except Exception as e:
        return False, f"Помилка виклику процесу: {str(e)}", sim_dir

def _get_column_mapping(columns):
    cols_to_keep = {}
    for col in columns:
        if "Date/Time" in col:
            cols_to_keep[col] = "Datetime"
        elif "Outdoor Air Drybulb Temperature" in col:
            cols_to_keep[col] = "T_out (C)"
        elif "Wind Speed" in col:
            cols_to_keep[col] = "Wind Speed (m/s)"
        elif "Zone Mean Air Temperature" in col:
            cols_to_keep[col] = "T_in (C)"
        elif "Zone Air CO2 Concentration" in col:
            cols_to_keep[col] = "CO2 (ppm)"
        elif "Generic Air Contaminant" in col:
            cols_to_keep[col] = "Generic Contaminant"
        elif (
            "Zone Air Infiltration Volume" in col
            or "Infiltration Standard Density Volume Flow Rate" in col
        ):
            cols_to_keep[col] = "Infiltration Volume (m3)"
        elif "Infiltration Sensible Heat Loss" in col:
            cols_to_keep[col] = "Infiltration Heat Loss (J)"
        elif "Fan Electricity Energy" in col or (
            "Electric" in col and "Fan" in col
        ):
            cols_to_keep[col] = "Fan Energy (J)"

```

```

elif "Electricity:Facility" in col:
    cols_to_keep[col] = "Total Electricity (J)"
elif "Sensible Heating Energy" in col and (
    "Ideal Loads" in col or "System" in col
):
    cols_to_keep[col] = "Heating Energy (J)"
elif "Sensible Cooling Energy" in col and (
    "Ideal Loads" in col or "System" in col
):
    cols_to_keep[col] = "Cooling Energy (J)"
elif "System Node Temperature" in col and "ERV_SA_OUTLET" in col.upper():
    cols_to_keep[col] = "T_supply (C)"
elif "Ventilation Air Changes per Hour" in col:
    cols_to_keep[col] = "Ventilation ACH"
elif "Heat Exchanger Sensible Heating Energy" in col:
    cols_to_keep[col] = "Heat Recovery (J)"
return cols_to_keep

def get_results(csv_path: str) -> pd.DataFrame:
    """
    Parses the eplusout.csv file to extract specific columns.
    """
    try:
        df = pd.read_csv(csv_path)
        # We look for partial matches to be robust against E+ version naming
        changes
        cols_to_keep = _get_column_mapping(df.columns)

        if not cols_to_keep:
            return pd.DataFrame()

        df_filtered = df[list(cols_to_keep.keys())].copy()

        # Гарантуємо унікальність нових назв колонок (оскільки вентиляторів тепер
        два)
        seen = {}
        unique_cols_to_keep = {}
        for original_col, target_col in cols_to_keep.items():
            if target_col in seen:
                seen[target_col] += 1
                unique_cols_to_keep[original_col] = f"{target_col}
{seen[target_col]}"
            else:
                seen[target_col] = 1
                unique_cols_to_keep[original_col] = target_col

        df_filtered.rename(columns=unique_cols_to_keep, inplace=True)

        # Обробка Дати/Часу для осі X
        if "Datetime" in df_filtered.columns:
            s_dt = df_filtered["Datetime"].str.strip()
            mask_24 = s_dt.str.endswith("24:00:00")
            s_dt = s_dt.str.replace("24:00:00", "00:00:00")

            # Додаємо рік (2026, як у IDF RunPeriod) та парсимо
            df_filtered["Datetime"] = pd.to_datetime(
                "2026/" + s_dt, errors="coerce", format="mixed"
            )
            df_filtered.loc[mask_24, "Datetime"] += pd.Timedelta(days=1)

        return df_filtered

```

```

except Exception as e:
    print(f"Помилка парсингу результатів: {e}")
    return pd.DataFrame()

class SimulationRunner:
    def __init__(
        self, base_project_json: str, eplus_exe: str = None, epw_path: str = None
    ):
        self.base_project_json = base_project_json
        self.eplus_exe = eplus_exe
        self.epw_path = epw_path
        self.dataset_path = "data/training_dataset.csv"

    def generate_random_params(self):
        import json

        data = json.loads(self.base_project_json)
        if "settings" not in data:
            data["settings"] = {}

        data["settings"]["wall_thickness"] = round(random.uniform(0.1, 0.4), 2)

        # Вибір випадкового стрес-сценарію
        scenario = random.choices(
            ["normal", "crowd", "blackout", "extreme_pollution", "holiday_empty"],
            weights=[0.3, 0.25, 0.15, 0.2, 0.1]
        )[0]

        if scenario == "crowd":
            data["settings"]["occupants"] = random.randint(20, 50)
            data["settings"]["recuperator_efficiency"] =
round(random.uniform(50.0, 90.0), 1)
            data["settings"]["recuperator_max_flow_m3_h"] = random.choice([100.0,
250.0, 300.0, 600.0])
            data["settings"]["heater_power"] = random.randint(0, 3000)
            data["settings"]["soldering_active"] = random.choice([0.0, 1.0])
            data["settings"]["printer_active"] = random.choice([0.0, 1.0])
        elif scenario == "blackout":
            data["settings"]["occupants"] = random.randint(5, 15)
            data["settings"]["recuperator_efficiency"] = 0.0
            data["settings"]["recuperator_max_flow_m3_h"] = 0.0 # Вентиляція
фізично вимкнена
            data["settings"]["heater_power"] = 0
            data["settings"]["soldering_active"] = 0.0
            data["settings"]["printer_active"] = 0.0
        elif scenario == "extreme_pollution":
            data["settings"]["occupants"] = random.randint(5, 15)
            data["settings"]["recuperator_efficiency"] =
round(random.uniform(50.0, 90.0), 1)
            data["settings"]["recuperator_max_flow_m3_h"] = random.choice([100.0,
250.0, 300.0, 600.0])
            data["settings"]["heater_power"] = random.randint(0, 3000)
            data["settings"]["soldering_active"] = random.uniform(5.0, 10.0) #
Екстремальні викиди
            data["settings"]["printer_active"] = random.uniform(5.0, 10.0)
        elif scenario == "holiday_empty":
            data["settings"]["occupants"] = 0
            data["settings"]["recuperator_efficiency"] = 0.0

```

```

        data["settings"]["recuperator_max_flow_m3_h"] = 0.0 # Економія енергії
        data["settings"]["heater_power"] = 0
        data["settings"]["soldering_active"] = 0.0
        data["settings"]["printer_active"] = 0.0
    else: # normal
        data["settings"]["occupants"] = random.randint(1, 15)
        data["settings"]["recuperator_efficiency"] =
round(random.uniform(50.0, 90.0), 1)
        data["settings"]["recuperator_max_flow_m3_h"] = random.choice([100.0,
250.0, 300.0, 600.0])
        data["settings"]["heater_power"] = random.randint(0, 3000)
        data["settings"]["soldering_active"] = random.choice([0.0, 1.0])
        data["settings"]["printer_active"] = random.choice([0.0, 1.0])

    return json.dumps(data)

def extract_features_targets(
    self, df_res: pd.DataFrame, proj_data: dict
) -> pd.DataFrame:
    sett = proj_data.get("settings", {})
    geom = proj_data.get("geometry", {})

    df = df_res.copy()

    # 1. Створення часових та календарних фічей (Calendar Features)
    if "Datetime" in df.columns:
        # Сортювання для впевненості у правильному хронологічному порядку
        df = df.sort_values("Datetime")
        df["Hour"] = df["Datetime"].dt.hour
        df["DayOfWeek"] = df["Datetime"].dt.dayofweek
        # Робочі години: з 8 ранку до 18 вечора включно
        df["Is_Working_Hour"] = ((df["Hour"] >= 8) & (df["Hour"] <=
18)).astype(int)

    # 2. Створення віконних/Lag (запізнілих) фічей (Lag Features)
    # Припускаємо, що датафрейм є послідовним в часі. EnergyPlus рахує з
кроком 1 година,
    # тому shift(1)=1 годину тому, shift(2)=2 години тому.
    # 2-годинне вікно важливе для теплової інерції стін: температура реагує на
зовнішній вплив
    # з запізненням 1-2 години.
    if "T_out (C)" in df.columns:
        df["T_out_lag_1"] = df["T_out (C)"].shift(1)

    if "T_in (C)" in df.columns:
        df["T_in_lag_1"] = df["T_in (C)"].shift(1)
        df["T_in_lag_2"] = df["T_in (C)"].shift(2) # 2 години тому

    if "CO2 (ppm)" in df.columns:
        df["CO2_lag_1"] = df["CO2 (ppm)"].shift(1)
        df["CO2_lag_2"] = df["CO2 (ppm)"].shift(2) # 2 години тому
        # Тренд CO2: швидкість зміни концентрації за вікно у 2 години
        df["CO2_trend"] = df["CO2 (ppm)"] - df["CO2_lag_1"]

    # 3. Додавання статичних параметрів проекту до кожного рядка як незалежних
ознак
    l = geom.get("L", 0)
    w = geom.get("W", 0)
    h = geom.get("H", 0)
    lc = geom.get("L_cut", 0)
    wc = geom.get("W_cut", 0)

```

```

# Правильний об'єм Г-подібної кімнати: (загальний прямокутник - виріз) *
висота
df["Volume_m3"] = (l * w - lc * wc) * h
df["Wall_Thickness"] = sett.get("wall_thickness", 0.0)
df["People_Count"] = sett.get("occupants", 0)
df["Soldering_Active"] = sett.get("soldering_active", 0.0)
df["Printer_Active"] = sett.get("printer_active", 0.0)
df["Heater_Power"] = sett.get("heater_power", 0)
df["Recuperator_Efficiency"] = sett.get("recuperator_efficiency", 0.0)

# 4. Температура припливного повітря після рекуперації – прямий фізичний
сигнал для передбачення T_in.
# EnergyPlus виводить T_supply з ERV SA Outlet, але колонка може бути
відсутня в сценаріях "blackout" / "holiday_empty".
if "T_supply (C)" in df.columns:
    df["T_supply_lag_1"] = df["T_supply (C)"].shift(1) # запізнена
температура припливу
else:
    df["T_supply_lag_1"] = 0.0 # вентиляція відключена (сценарій без ERV)

# 5. Видалення рядків з NaN (які утворились через зсув shift)
# Перші два рядки симуляції будуть видалені (через lag_1 i lag_2).
df.dropna(inplace=True)

return df

def run_batch(self, n_simulations=50):
    import json
    import tempfile

    from simulation import idf_bridge

    results_dfs = []
    for i in range(n_simulations):
        json_project = self.generate_random_params()
        idf_data = idf_bridge.generate_idf_structure(json_project)

        with tempfile.TemporaryDirectory() as tmpdir:
            success, msg, _ = run_simulation(
                idf_data, self.eplus_exe, self.epw_path,
                sim_dir_override=tmpdir
            )

            if success:
                df = get_results(msg)
                if not df.empty:
                    df_target = self.extract_features_targets(
                        df, json.loads(json_project)
                    )
                    if not df_target.empty:
                        results_dfs.append(df_target)

    if results_dfs:
        # Об'єднуємо всі датафрейми з різних симуляцій в один великий
        (вертикально)
        df_log = pd.concat(results_dfs, ignore_index=True)

        file_exists = os.path.exists(self.dataset_path)
        is_empty = file_exists and os.path.getsize(self.dataset_path) == 0

        # Забезпечуємо стабільний порядок колонок (алфавітний або фіксований)

```

```

# Це критично для режиму append (mode='a')
df_log = df_log.reindex(sorted(df_log.columns), axis=1)

os.makedirs(os.path.dirname(self.dataset_path), exist_ok=True)
if not file_exists or is_empty:
    df_log.to_csv(self.dataset_path, index=False)
else:
    # Читаємо існуючі колонки для перевірки сумісності
    try:
        df_existing = pd.read_csv(self.dataset_path, nrows=0)
        df_log = df_log.reindex(columns=df_existing.columns)
    except Exception:
        pass
    df_log.to_csv(self.dataset_path, mode="a", header=False,
index=False)
    return True, len(df_log)
return False, 0

```

ДОДАТОК Б

Програмний додаток для підбору рекуператора методами машинного навчання на основі цифрового двійника приміщення

Презентація

Слайдів 14

Київ – 2026



Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ПРОГРАМНИЙ ДОДАТОК ДЛЯ ПІДБОРУ РЕКУПЕРАТОРА МЕТОДАМИ МАШИННОГО НАВЧАННЯ НА ОСНОВІ ЦИФРОВОГО ДВІЙНИКА ПРИМІЩЕННЯ

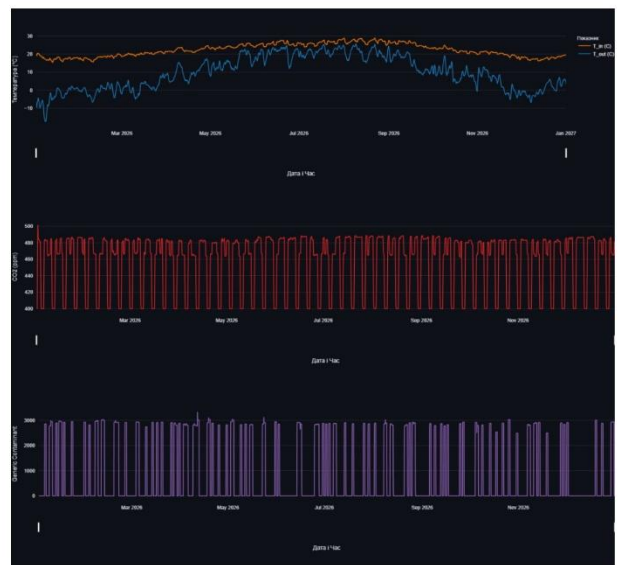
виконав: студент групи ТВ-21 Гранківський Дмитро Петрович

керівник: асистент Швайко Валерій Григорович

2026

АКТУАЛЬНІСТЬ ТЕМИ

- Проблема:** Необхідність якісного підбору приладів контролю мікроклімату.
- Специфіка:** Нелінійні сплески CO₂ та пилу PM_{2.5} в інженерних лабораторіях.
- Недолік аналогів:** Статичні розрахунки не враховують експлуатаційну динаміку.
- Мета:** Автоматизація підбору рекуператора через симуляційне тестування кандидатів у контурі цифрового двійника.



ПОСТАНОВКА ЗАДАЧІ

Метою роботи є розробка інтелектуального програмного комплексу «HIVanalyzer» для автоматизованого підбору оптимальних моделей рекуператорів на основі цифрового двійника приміщення з використанням машинного навчання.



АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Проблеми існуючих рішень

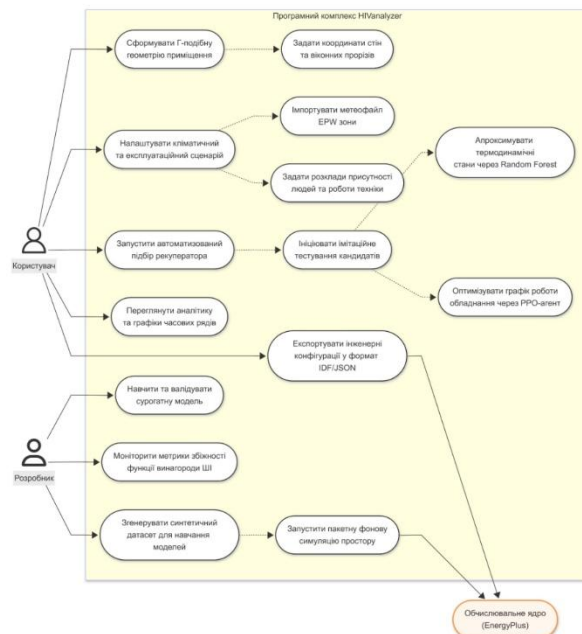
- Статичність розрахунків: Ігнорування реальної динаміки експлуатації приміщення.
- Обчислювальний бар'єр: Низька швидкодія класичних інженерних симуляторів.
- Відсутність автоматизації: Існуюче програмне забезпечення вимагає ручного підбору параметрів і не має вбудованих інтелектуальних контурів ранжування.

Приклади аналогів



ПРОЄКТУВАННЯ СИСТЕМИ

Функціонал комплексу забезпечує інтерактивне створення геометрії приміщення, налаштування експлуатаційних розкладів та автоматичний підбір обладнання.

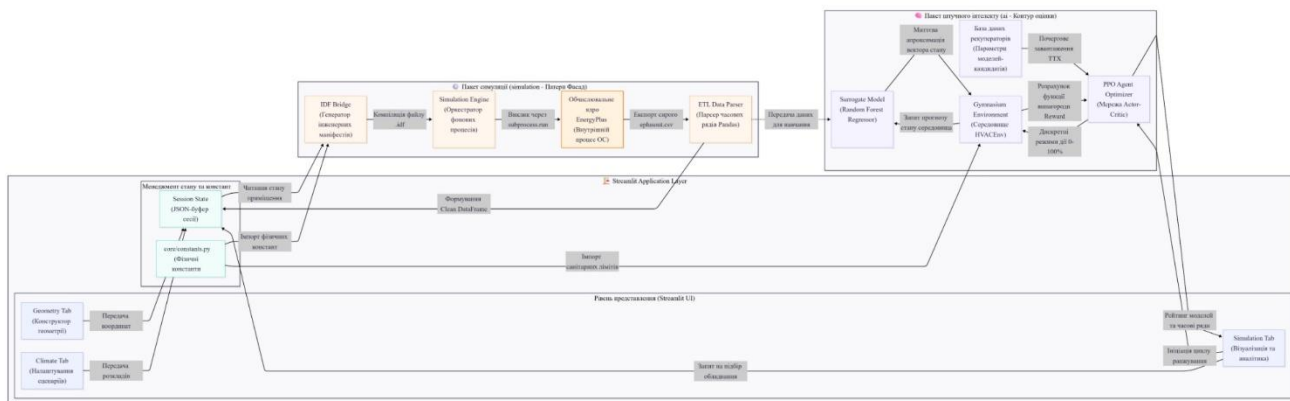


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

5/14

АРХІТЕКТУРА СИСТЕМИ

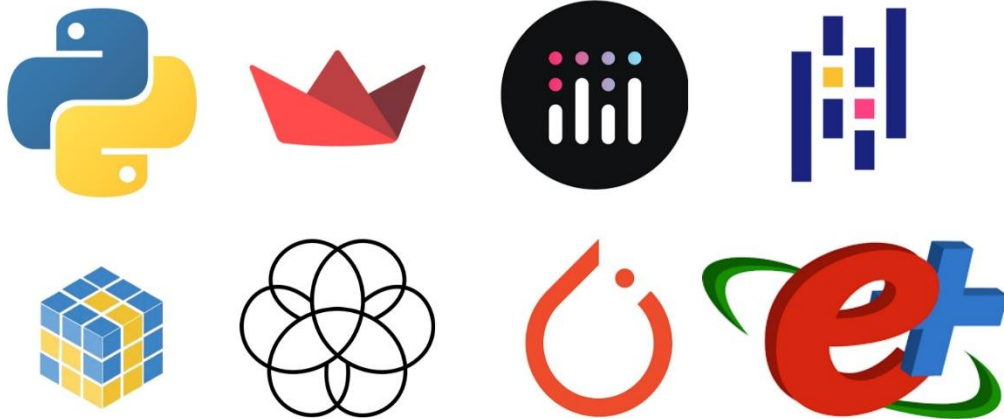
Діаграма компонентів



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

6/14

ВИКОРИСТАНІ ТЕХНОЛОГІЇ



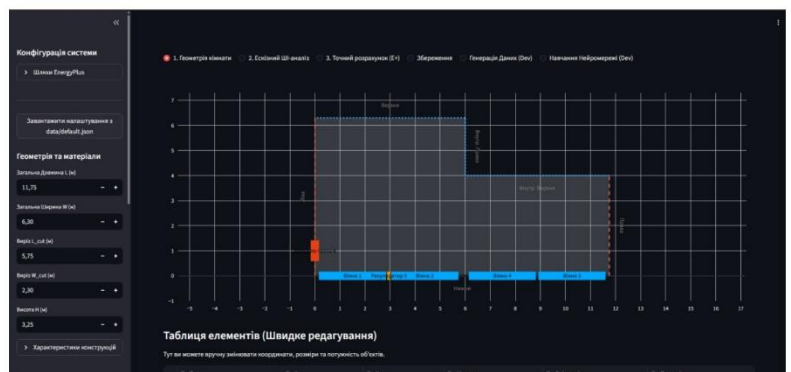
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

7/14

РЕАЛІЗАЦІЯ СИСТЕМИ

Інтерфейс користувача

Графічний інтерфейс користувача реалізовано як односторінковий веб-застосунок із реактивним перемальовуванням елементів при зміні вхідних даних. Інтерактивна вкладка геометричного конструктора забезпечує візуальне проектування Г-подібних кімнат та автоматичну серіалізацію координат у обмінний формат JSON.



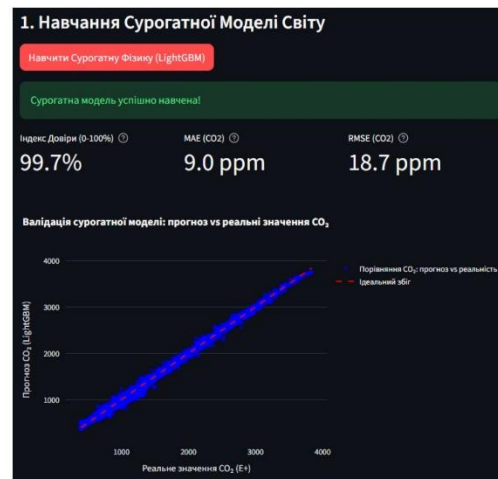
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

8/14

РЕАЛІЗАЦІЯ СИСТЕМИ

Сурогатна модель

Побудована сурогатна модель на базі алгоритму LightGBM виступає швидким математичним апроксиматором складних фізичних процесів приміщення.



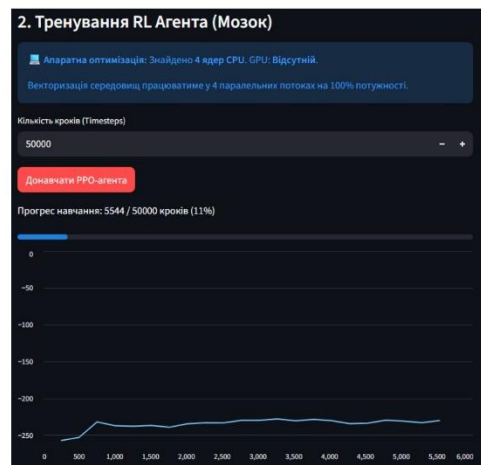
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

9/14

РЕАЛІЗАЦІЯ СИСТЕМИ

PPO модель

Алгоритм навчання з підкріпленням PPO інтегровано як внутрішній двигун оцінки та ранжування обладнання. Штучний інтелект імітує оптимальну роботу кожного рекуператора у дискретному просторі дій від 0% до 100% потужності.



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

10/14

ТЕСТУВАННЯ СИСТЕМИ

Тестове покриття

Автоматизоване модульне тестування за допомогою фреймворку PyTest забезпечує перевірку ізольованої логіки інженерних та ІІІ-модулів програми.

Coverage report: 67%				
Files Functions Classes				
coverage.py v7.13.5, created at 2026-05-10 15:43 +0300				
File	statements	missing	excluded	coverage
ai_init_.py	0	0	0	100%
ai_ai_engine.py	60	12	0	80%
ai_ml_surrogate.py	44	4	0	91%
ai_rl_agent.py	170	42	0	75%
ai_rl_environment.py	127	10	0	92%
ai_train_agent.py	22	14	0	36%
core_init_.py	0	0	0	100%
core_constants.py	4	0	0	100%
core_extract.py	53	7	0	87%
simulation_init_.py	0	0	0	100%
simulation_idf_bridge.py	554	91	0	84%

Summary

34 tests took 00:00:23.

(Un)check the boxes to filter the results:

☐ 0 Failed, ☒ 34 Passed, ☐ 0 Skipped, ☐ 0 Expected failures, ☐ 0 Unexpected passes, ☐ 0 Errors, ☐ 0 Reruns, ☐ 0 Retried.

Result	Test
Passed	tests/test_ai_engine.py::test_generate_xai_explanation
Passed	tests/test_ai_engine.py::test_simulate_24h_fallback_math
Passed	tests/test_ai_env.py::test_hvac_env_initialization
Passed	tests/test_ai_env.py::test_reward_calculation_extreme_co2

```
=====
Complexity analysis
=====
ai\ai_engine.py
  F 27:0 simulate_24h - B (7)
  F 111:0 generate_xai_explanation - B (6)
  F 13:0 load_ai_model - A (2)
  F 22:0 clear_model_cache - A (1)
ui\tabs\data_gen_tab.py
  F 24:0 render - B (8)
  F 15:0 load_end_analyze_dataset - A (4)
ui\tabs\geometry_tab.py
  F 0:0 render - A (2)
ui\tabs\save_tab.py
  F 9:0 render - A (1)

74 blocks (classes, functions, methods) analyzed.
Average complexity: B (5.95945945945946)
=====
Maintainability Index
=====
ai\ai_engine.py - A
ai\ml_surrogate.py - A
ai\rl_agent.py - A
ai\rl_environment.py - A
ai\train_agent.py - A
ai\_init_.py - A
simulation\idf_bridge.py - A
simulation\simulation_engine.py - A
simulation\validator.py - A
simulation\_init_.py - A
core\constants.py - A
core\extract.py - A
core\_init_.py - A
ui\designer.py - A
ui\_init_.py - A
ui\tabs\advisor_tab.py - A
ui\tabs\ai_lab_tab.py - A
ui\tabs\simulate_tab.py - A
```



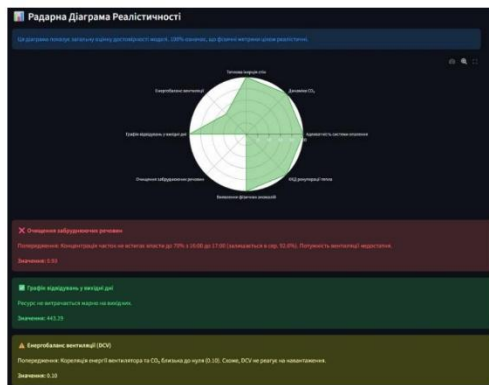
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

11/14

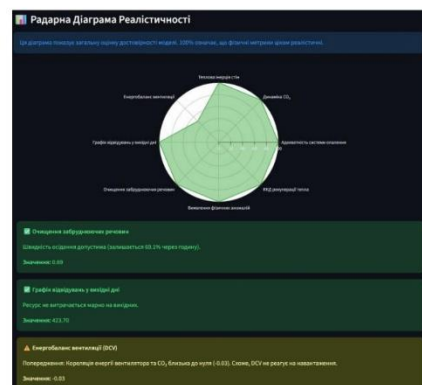
ТЕСТУВАННЯ СИСТЕМИ

Результати симуляцій

До підбору обладнання



Після підбору обладнання



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

12/14

ВИСНОВКИ

Розроблено: Програмний комплекс підтримки прийняття рішень «HIVanalyzer».

Реалізовано: Графічний конструктор для формалізації нетипової геометрії приміщень.

Побудовано: Швидкий сурогат на Random Forest з точністю апроксимації 99.7%.

Створено: Внутрішній імітаційний RL-контур оцінювання та ранжування моделей обладнання.

Протестовано: Надійні програмні модулі із показником unit-покриття коду до 67%.



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

13/14



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

—
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"