МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ ТА БІОТЕХНОЛОГІЙ ІМ. С.З.ГЖИЦЬКОГО
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського) рівня вищої освіти

на тему: **«Інтелектуальна система управління теплицею на основі генетичного алгоритму»**

Виконав: студент групи Іт-62

Спеціальності 126 «Інформаційні системи та технології»
(шифр і назва)
Буцяк Максим Васильович
(Прізвище та ініціали)

Керівник: д.т.н., професор Тригуба А.М.
(Прізвище та ініціали)

Рецензент: к.т.н., доцент Шарибура А.О.
(Прізвище та ініціали)

ДУБЛЯНИ-2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Другий (магістерський) рівень вищої освіти
Спеціальність 126 «Інформаційні системи та технології»

«ЗАТВЕРДЖУЮ»

Завідувач кафедри_____

д.т.н., проф. А.М. Тригуба
«____» _____2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу студенту

Буцяку Максиму Васильовичу

1. Тема роботи: «Інтелектуальна система управління теплицею на основі генетичного алгоритму»

Керівник роботи Тригуба Анатолій Миколайович, професор
затверджені наказом по університету від 28.02.2025 року № 140/к-с.

2. Строк подання студентом роботи 10.12.2025 р.

3. Вихідні дані до роботи: *дані моніторингу параметрів мікроклімату теплиці; характеристики технічних систем вентиляції, опалення та поливу; алгоритмічні моделі генетичної оптимізації; програмні засоби реалізації інтелектуальних систем управління.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)
Вступ.
Аналіз стану оптимізації мікроклімату теплиць та використання інтелектуальних систем управління.
Розробка концепції інтелектуальної системи управління теплицею на основі генетичного алгоритму.
Результати створення та тестування інтелектуальної системи управління мікрокліматом теплиці.
Охорона праці та безпека у надзвичайних ситуаціях.
Визначення економічної ефективності.
Висновки та пропозиції.

Список використаної літератури.

5. Перелік ілюстраційного матеріалу (з точним зазначенням обов»язкових слайдів): *аналіз стану прогнозування конкурсного балу соціальних проєктів та використання нейромереж; розробка концепції нейромережевої моделі для прогнозування конкурсного балу та підготовка даних; результати створення нейромережевих моделей для прогнозування конкурсного балу соціальних проєктів розвитку громад; економічна ефективність.*

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3, 5	*Тригуба А.М., зав. кафедри інформаційних технологій*		
4	*Городецький І.М., доцент кафедри інженерної механіки*		

7. Дата видачі завдання 28 лютого 2025 р.

Календарний план

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів роботи	Примітка
1	*Написання першого розділу*	*28.02-20.03.25*	
2	*Виконання другого розділу та аркушів ілюстраційного матеріалу до нього*	*21.03-14.06.25*	
3.	*Виконання третього розділу та аркушів ілюстраційного матеріалу до нього*	*15.06.25-10.07.25*	
4.	*Написання розділу «Охорона праці та безпека у надзвичайних ситуаціях»*	*11.07-31.08.25*	
5.	*Оцінення ефективності запропонованої системи*	*01.09-31.10.25*	
6.	*Завершення оформлення розрахунково-пояснювальної записки та аркушів ілюстраційного матеріалу*	*01-30.11.25*	
7.	*Завершення роботи вцілому*	*01-10.12.25*	

Студент _____ Буцяк М.В.
(підпис)

Керівник роботи _____ Тригуба А.М.
(підпис)

УДК 004.89:631.67

Інтелектуальна система управління теплицею на основі генетичного алгоритму.

Буцяк М.В. Кафедра інформаційних технологій – Дубляни, ЛНУВМ, 2025.

Кваліфікаційна робота: 102 с. текст. част., 19 рис., 9 табл., 15 арк. ілюстраційного матеріалу, 35 джерел.

Подано особливості та доцільність оптимізації мікроклімату теплиць у сучасному агровиробництві. Проаналізовано параметри мікроклімату, що впливають на продуктивність рослин. Розглянуто використання генетичних алгоритмів як ефективного інструменту оптимізації процесів у тепличному господарстві. Виконано аналіз наукових праць і подано напрями сучасних досліджень у галузі автоматизованих систем керування мікрокліматом. Сформульовано мету, завдання та етапи виконання кваліфікаційної роботи.

Розроблено концепцію побудови інтелектуальної системи управління теплицею на основі генетичного алгоритму. Реалізовано удосконалений генетичний алгоритм з адаптивними стратегіями відбору, схрещування та мутації для енергетичної системи теплиці. Створено апаратно-програмний комплекс управління та програмні модулі «Python Genetic Algorithm Module» і «Web Dashboard / LCD Display» для моніторингу й оптимізації мікроклімату. Наведено результати аналізу збіжності алгоритму та ефективності системи керування.

Виконано аналіз умов праці під час монтажу системи, розроблено логічно-імітаційну модель травматизму та визначено заходи безпеки у надзвичайних ситуаціях. Проведено економічну оцінку ефективності впровадження інтелектуальної системи управління теплицею на основі генетичного алгоритму.

Ключові слова: мікроклімат, теплиця, інтелектуальна система, генетичний алгоритм, оптимізація, управління, збіжність, енергетична ефективність.

ЗМІСТ

ВСТУП

Сучасне сільське господарство перебуває у стані активної цифрової трансформації, що зумовлюється глобальними викликами, пов’язаними з кліматичними змінами, зростанням населення та потребою у збільшенні обсягів продовольства при одночасному раціональному використанні ресурсів. Тепличні господарства відіграють ключову роль у забезпеченні безперервного виробництва овочевих і квіткових культур незалежно від сезону. Проте ефективність їх функціонування визначається не лише технологіями вирощування, а й рівнем автоматизації процесів контролю мікроклімату. Оптимальне регулювання параметрів середовища, таких як температура, вологість, концентрація вуглекислого газу та рівень освітленості, є складним завданням, оскільки потребує врахування динамічних змін зовнішніх факторів, внутрішніх особливостей теплиці та біологічних потреб рослин [25].

Традиційні методи управління теплицями, засновані на класичних ПІД-регуляторах, мають низку обмежень, оскільки вони ефективні лише за стабільних умов і не здатні швидко адаптуватися до раптових змін довкілля. У свою чергу, інтелектуальні методи управління, зокрема ті, що використовують алгоритми оптимізації та машинного навчання, дозволяють підвищити точність регулювання та зменшити енергетичні витрати. Останніми роками значну увагу дослідників привертають генетичні алгоритми, які завдяки своїй здатності до глобального пошуку оптимальних рішень виявили високу ефективність у задачах керування нелінійними та багатофакторними системами [15].

У дослідженні, що взято за основу даної роботи [7], продемонстровано використання методів штучного інтелекту для оптимізації тепличних процесів. Автори показали, що поєднання математичного моделювання мікроклімату з алгоритмами еволюційного пошуку дозволяє досягати стабільного контролю над параметрами середовища навіть за умов стохастичних збурень. Це підтверджує доцільність застосування генетичного алгоритму для створення інтелектуальної системи управління теплицею, здатної адаптуватися до

мінливих умов та забезпечувати баланс між енергоефективністю і продуктивністю.

Метою роботи є розроблення інтелектуальної системи управління теплицею, яка забезпечує підтримку оптимальних умов для росту рослин завдяки використанню генетичного алгоритму для оптимізації керуючих дій. Для досягнення цієї мети передбачено аналіз існуючих підходів до автоматизованого управління теплицями, дослідження можливостей генетичного алгоритму у розв'язанні задачі багатопараметричної оптимізації, проєктування архітектури інтелектуальної системи та створення програмного прототипу, здатного демонструвати ефективність запропонованого підходу.

Об'єкт дослідження – система управління мікрокліматом теплиці, що охоплює апаратно-програмний комплекс збору, обробки та регулювання параметрів внутрішнього середовища (температури, вологості, освітленості та концентрації CO_2) з метою забезпечення оптимальних умов для росту рослин.

Предмет дослідження – методи та алгоритми оптимізації процесів регулювання мікроклімату в теплиці, зокрема застосування генетичного алгоритму для визначення оптимальних керуючих впливів на системи вентиляції, зволоження, обігріву та освітлення з метою підтримки стабільних умов вирощування рослин і підвищення ефективності використання енергетичних та водних ресурсів.

Засоби дослідження – теоретичні методи системного аналізу та математичного моделювання, алгоритмічні засоби оптимізації на основі генетичного алгоритму, інструменти програмної реалізації (Python, бібліотеки для еволюційних обчислень та симуляційних експериментів), а також апаратні компоненти системи моніторингу (датчики температури, вологості, освітленості, рівня CO_2) та виконавчі механізми тепличного обладнання (системи вентиляції, зволоження, обігріву, освітлення).

Розділ 1.
СТАН ДОСЛІДЖЕНЬ ТА ПРАКТИКА УПРАВЛІННЯ МІКРОКЛІМАТОМ ТЕПЛИЦЬ

1.1. Актуальність проблеми оптимізації мікроклімату теплиць у сучасному агровиробництві

Теперішнє агровиробництво стикається з подвійним викликом – необхідністю забезпечення стабільного виробництва продуктів харчування та одночасним зниженням витрат ресурсів у зв'язку з глобальними кліматичними змінами. Тепличні комплекси відіграють особливу роль у цьому процесі, адже вони дозволяють отримувати врожай незалежно від сезону. Проте функціонування таких споруд є ресурсомістким, а підтримка стабільного мікроклімату часто потребує значних витрат енергії та води. Саме тому оптимізація мікроклімату теплиць постає однією з ключових наукових і практичних проблем сучасного сільського господарства [25].

Основними параметрами, що визначають ріст і розвиток рослин, є температура, відносна вологість, рівень освітленості та концентрація вуглекислого газу. Відхилення хоча б одного з них від оптимального діапазону призводить до уповільнення фотосинтезу та зниження врожайності. У практиці агротехнологій це означає додаткові витрати, пов'язані з низькою ефективністю використання ресурсів. На рисунку 1.1 представлено взаємозв'язок між основними параметрами мікроклімату та продуктивністю тепличних культур.

У роботі [23] запропоновано комплексний підхід для оптимізації стратегії управління продуктивністю теплиці, забезпечення оптимальних умов вирощування та полегшення контролю параметрів навколишнього середовища. Основні моменти такі:

1. Геометрія теплиці включає її структуру та функціонування для досягнення енергоефективності, максимізації росту рослин та мінімізації впливу на навколишнє середовище. Моделювання теплиць може скористатися

перевагами різних методів та стратегій для досягнення цих цілей. Одним із методів оптимізації моделювання теплиць є утеплення фундаменту для запобігання втратам тепла через підлогу теплиці, щоб забезпечити теплоізоляцію та підтримку конструкції теплиці.

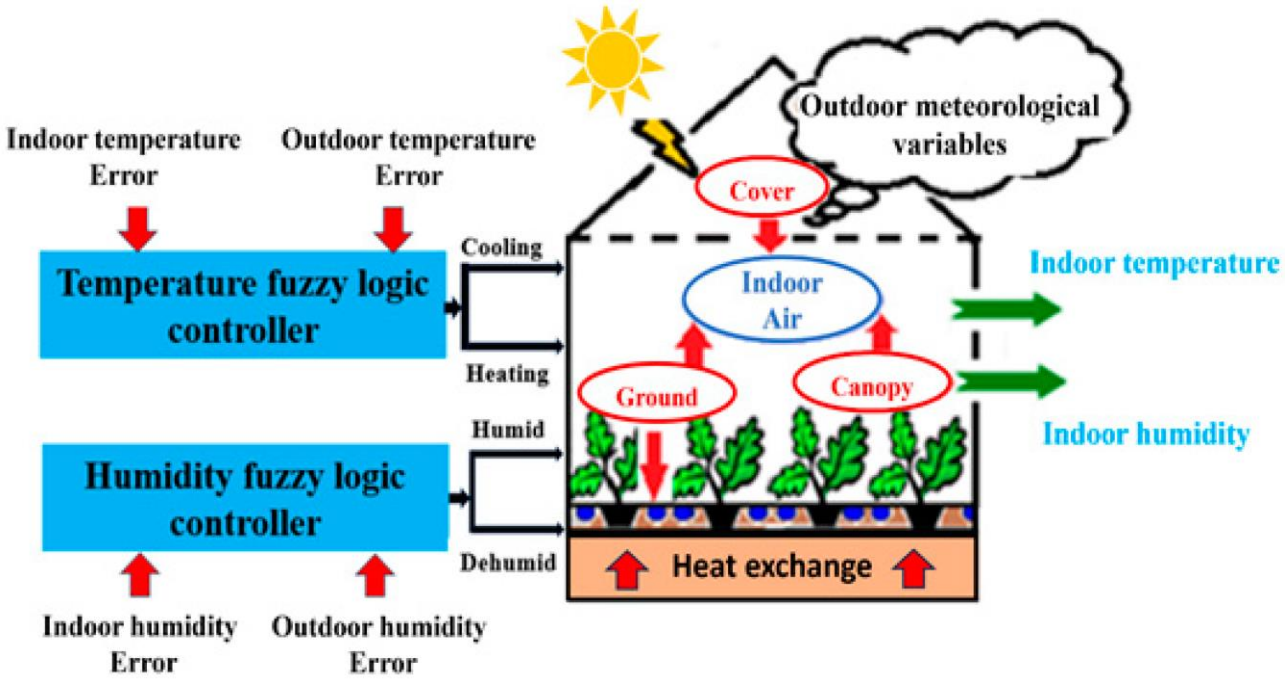

Рисунок 1.1 – Концептуальна схема тестованого SIG та його елементів керування [23]

2. Були досліджені механізми теплопередачі в запропонованій тепличній системі (SIG) для створення ідеального середовища в приміщенні для вирощування. Зокрема, вплив температури крони, покриву та температури ґрунту в тепличній системі може суттєво впливати на клімат у приміщенні та ріст рослин. Ці фактори включені в математичну модель SIG для створення ідеального тепличного середовища, що включає ретельний баланс для задоволення конкретних потреб рослин.

3. Підхід до керування базується на впровадженні приладів для моніторингу теплових параметрів SIG з використанням внутрішніх та зовнішніх датчиків. Контроль клімату в приміщенні досягається за допомогою інтелектуальних плавних регуляторів температури (FLC), які керують

виконавчими механізмами, враховуючи кліматичні коливання та їх вплив на технологію автоматизації (систему охолодження та опалення).

На рисунку 1.1 представлено SIG, що тестується, і він складається з таких частин:

1. Два FLC (сині блоки) для контролю температури та вологості SIG;
2. Теплична система (п’ятикутний блок), яка містить теплообмін між повітрям всередині приміщення та зовнішніми змінними (показано зеленими стрілками).

Для кількісної оцінки ефективності тепличного мікроклімату доцільно використовувати інтегральний показник, що враховує відхилення від нормативних значень. Його можна подати у вигляді функції:

$$I = \sum_{i=1}^{n} w_i \cdot \frac{|x_i - x_i^{opt}|}{x_i^{opt}}, \qquad (1.1)$$

де I – інтегральний показник відхилення мікроклімату; x_i – фактичне значення параметра; x_i^{opt} – оптимальне значення; w_i – ваговий коефіцієнт значущості параметра; n – кількість параметрів.

Чим меншим є значення I, тим більш стабільним та ефективним є мікроклімат теплиці. Така модель дозволяє не тільки кількісно оцінити рівень відхилення, але й застосувати її як цільову функцію при оптимізації за допомогою алгоритмів штучного інтелекту.

З точки зору практичного агровиробництва, відхилення умов середовища від нормативів безпосередньо впливають на собівартість продукції. У таблиці 1 наведено узагальнені дані щодо впливу відхилень основних параметрів мікроклімату на врожайність томатів у теплицях на прикладі досліджень [7].

Оптимізація мікроклімату є також важливою у контексті енергоефективності. За даними сучасних досліджень, до 50 % енергоспоживання теплиць пов’язане із системами обігріву та вентиляції [13]. Використання інтелектуальних методів управління дозволяє знизити витрати енергії на 15…25%, що у масштабах промислових тепличних комплексів дає суттєвий економічний ефект. Водночас алгоритми оптимізації забезпечують

стабільність параметрів, що критично важливо для підвищення врожайності та якості продукції.

Таблиця 1.1 – Вплив параметрів мікроклімату на врожайність тепличних томатів

Параметр	Оптимальний діапазон	Відхилення від оптимуму	Зниження врожайності, %
Температура, °C	22…26	+5	12
Вологість, %	60…70	–15	18
Освітленість, люкс	8 000…12000	–30 %	25
CO_2, мг/м³	800…1000	–50 %	20

Таким чином, актуальність проблеми оптимізації мікроклімату теплиць зумовлюється одночасною потребою у підвищенні ефективності агровиробництва та зниженні ресурсних витрат. Вирішення цього завдання можливе завдяки інтеграції автоматизованих систем моніторингу, інтелектуальних алгоритмів оптимізації та сучасних засобів управління, що і визначає наукову та практичну значимість дослідження.

1.2. Характеристика основних параметрів мікроклімату та їх значення для продуктивності рослин

Мікроклімат теплиці формується взаємодією комплексу параметрів, серед яких визначальними є температура повітря, відносна вологість, інтенсивність освітлення та концентрація вуглекислого газу. Вони створюють середовище для росту та розвитку рослин і визначають ефективність використання ресурсів. Будь-яке відхилення цих параметрів від оптимальних значень відображається

на врожайності та якості продукції. Саме тому управління мікрокліматом розглядається як основний чинник у підвищенні продуктивності тепличних культур [17].

Температура є головним фактором, що визначає інтенсивність біохімічних процесів у рослинах. Оптимальний діапазон для більшості овочевих культур становить 22–26 °C. При підвищенні температури вище 30 °C знижується ефективність фотосинтезу, а при зниженні нижче 18 °C відбувається уповільнення клітинного поділу та росту. Баланс теплової енергії у теплиці можна описати рівнянням:

$$Q_{in} - Q_{out} = \Delta Q, \qquad (1.2)$$

де Q_{in} – надходження тепла (сонячна радіація, обігрів); Q_{out} – втрати тепла (вентиляція, випромінювання); ΔQ – зміна теплової енергії всередині теплиці.

Обмін енергією, що відбувається в типовій теплиці, подано на рис. 1.2. Дослідження у роботі [11] стосується переважно високотехнологічних гідропонних теплиць комерційного масштабу, теплопередача між повітрям у приміщенні та землею вважається незначною через припущення, що використовується матеріал для підлоги з високими теплоізоляційними властивостями.

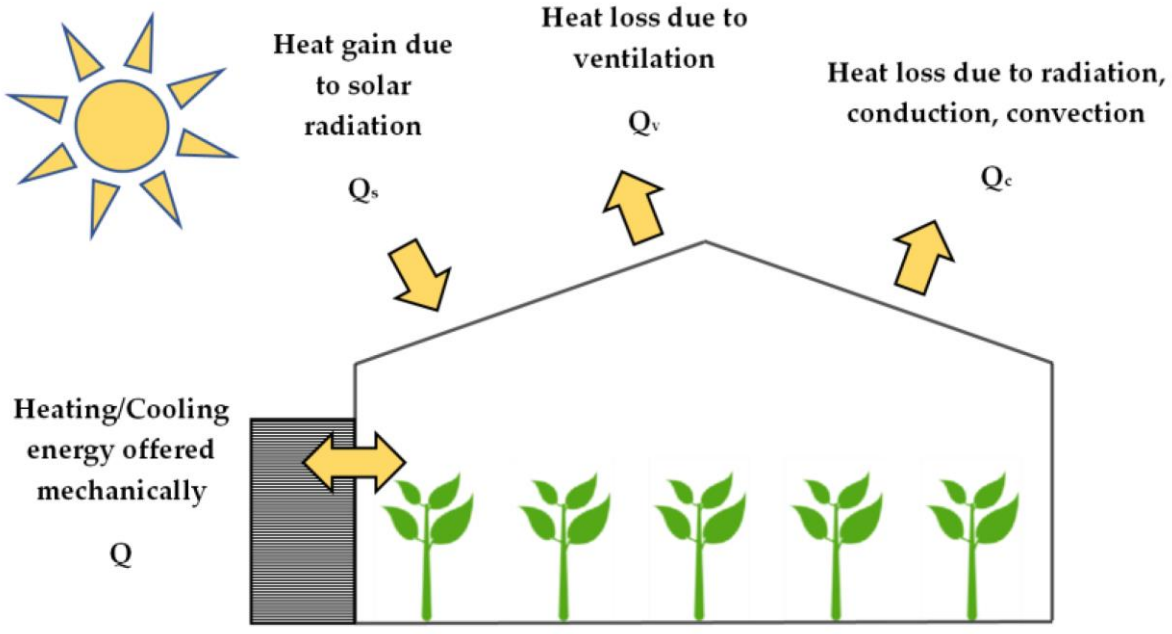

Рисунок 1.2 – Схема теплового балансу теплиці [11]

Енергетичний баланс у теплиці можна описати наступним рівнянням:

$$Q_s + Q = Q_c + Q_v, \quad (1.2)$$

де Q_c – приріст тепла від сонячної радіації, кВт·год/м² / д; Q – енергія, механічно подається до теплиці або видаляється з неї, кВт·год/м² / д; Q_c – втрати тепла внаслідок випромінювання, теплопровідності та конвекції, кВт·год/м² / д; Q_v – втрати тепла внаслідок вентиляції, кВт·год/м² / д.

Енергія Q, необхідна для підтримки температури повітря всередині теплиці в певних межах, або подається в теплицю ($Q>0$) у випадку опалення, або відводиться від неї ($Q<0$) у випадку охолодження. Щодо опалення теплиць, зазвичай використовуються котли або повітряні нагрівачі, тоді як для охолодження тепловими насосами часто застосовуються методи вентилятор-подушечка та туман-розпилювач [9].

У теплицях із сучасною ізоляцією вдається знизити втрати тепла до 30 % у порівнянні зі старими конструкціями, проте навіть у таких випадках підтримання температурного режиму залишається одним із найбільш енерговитратних процесів [10].

Відносна вологість повітря визначає інтенсивність випаровування вологи з поверхні листків і транспірацію. Для більшості культур оптимальні значення становлять 60–70 %. Висока вологість понад 80 % сприяє розвитку патогенів, тоді як низька вологість нижче 50 % призводить до закриття продихів і порушення газообміну. Для оцінки стану рослин застосовується показник дефіциту тиску насичення (VPD – Vapor Pressure Deficit):

$$VPD = e_s(T) - e_a, \quad (1.3)$$

де $e_s(T)$ – насичений парціальний тиск при температурі T; e_a – фактичний парціальний тиск водяної пари.

Незважаючи на негативний вплив дефіциту води на продуктивність рослин, рослини здатні реагувати на різний ступінь дефіциту води (рис. 1.3). Існує сильна кореляція між ростом рослин та доступністю води, оскільки збільшення клітин більше залежить від дефіциту води, ніж від поділу клітин. За

цих умов ріст рослин гальмується в результаті зниження розтяжності клітинної стінки та тургору. За сильних посух дихання також може зменшуватися, хоча збільшення дихання спостерігалося і при легкому стресі. Щоб впоратися з дефіцитом води, осмотична регуляція стресованих рослин підтримується за рахунок збільшення вмісту цукру в коренях і листках, і у рослин, які зазнали стресу від посухи в минулому, спостерігався відносно більший ріст коренів порівняно з пагонами [24].

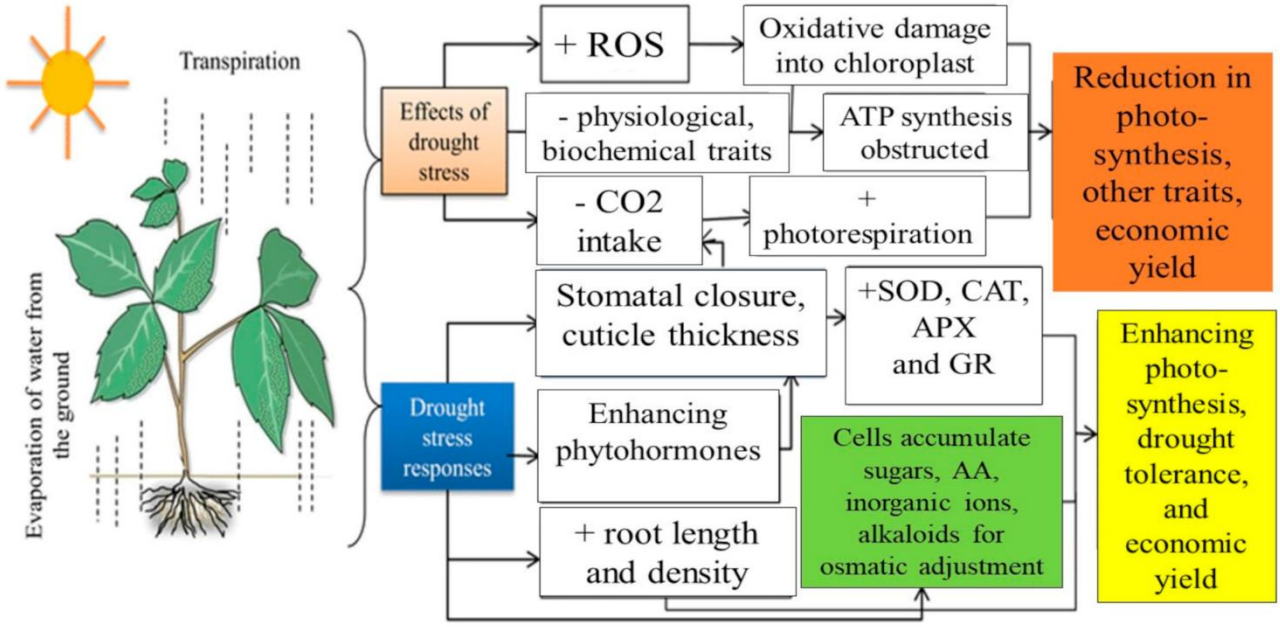

Рисунок 1.3 – Несприятливі наслідки та адаптація рослин до стресу посухи, модифіковані – означає зменшення; + означає збільшення [24]

Фактори навколишнього середовища, включаючи тривалість, інтенсивність та частоту посухи, характеристики ґрунту, умови та стадії росту, а також види рослин, сильно впливають на ступінь та тривалість симптомів, пов’язаних з посухою, у рослин [24].

Таким чином, регулювання вологості у теплиці тісно пов’язане з контролем вентиляції та системами зволоження.

Світло є головним джерелом енергії для фотосинтезу. Оптимальна освітленість для більшості овочевих культур у теплицях становить 8000…12000 люкс. При недостатній освітленості знижується швидкість фотосинтезу, а при

надлишку виникає ризик перегріву та пошкодження листя. Важливим є також спектральний склад світла: дослідження показали, що червоне (600–700 нм) і синє (400…500 нм) випромінювання найбільш ефективно стимулюють фотосинтетичну активність, тоді як зелена частина спектра має другорядний вплив [21].

Залежність фотосинтетичної продуктивності від освітленості описується функцією насичення:

$$P_n = \frac{P_{max} \cdot I}{I + K}, \qquad (1.4)$$

де P_n – чиста фотосинтетична продуктивність; P_{max} – максимальна швидкість фотосинтезу; I – інтенсивність світла; K – константа напівнасичення.

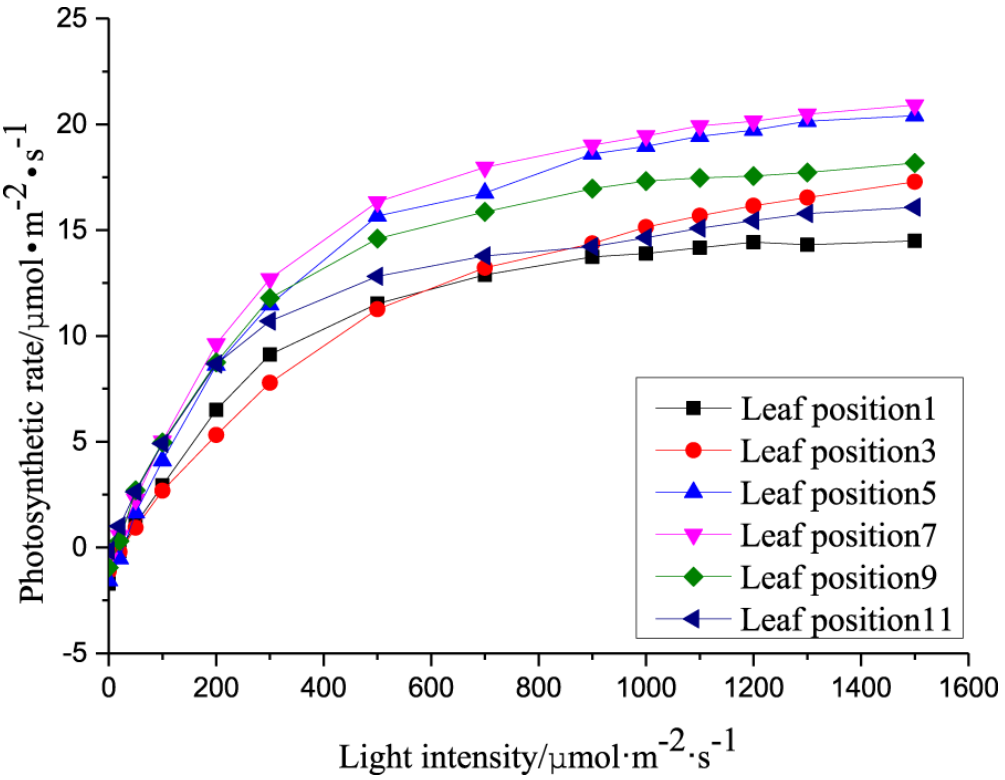

Рисунок 1.4 – Швидкість фотосинтезу в різних положеннях листка за температури 18 °С та концентрації CO^2 600 мкмоль·моль$^{-1}$ [35]

Спостерігалися очевидні відмінності у фотосинтетичній здатності між різними положеннями листка (рис. 1.4). Верхній листок (положення листка 1) має найгіршу фотосинтетичну здатність через неповний розвиток

фотосинтетичного апарату. Позиції листка 5…7 є функціональними позиціями листка, в яких фотосинтетичний апарат зрілий, а фотосинтетична здатність є відносно оптимальною порівняно з іншими положеннями листка в рослині. Хоча листки в положенні 11 старі, фотосинтетичний апарат зрілий та неушкоджений, а вміст хлорофілу відносно високий. Отже, фотосинтетична здатність у цьому положенні краща, ніж у верхнього листка. Однак, оскільки верхній листок сприяє поздовжньому росту завдяки своєму фототропізму, нове листя рослини відіграє важливу роль у рості та розвитку всієї рослини. Тому важливо точно побудувати модель прогнозування швидкості фотосинтезу цих новонароджених листків.

Таблиця 1.2 – Оптимальні параметри мікроклімату та їх вплив на продуктивність

Параметр	Оптимальний діапазон	Наслідки відхилень
Температура, °C	22–26	При >30 °C – зниження фотосинтезу; при <18 °C – уповільнення росту
Вологість, %	60–70	При >80 % – конденсат, патогени; при <50 % – водний стрес
Освітленість, lux	8 000–12 000	Недостатня – гальмування фотосинтезу; надлишок – перегрів
CO_2, ppm	800–1000	При <400 ppm – лімітований фотосинтез; при >1500 ppm – насичення

Вуглекислий газ є одним із лімітуючих факторів фотосинтезу. При природному рівні 400 ppm інтенсивність процесу є обмеженою. Підвищення концентрації до 800–1000 ppm може збільшити врожайність на 20–30 %, проте надмірні значення понад 1500 ppm практично не дають додаткового ефекту.

Для переходу від ppm до абсолютних одиниць використовують співвідношення:

$$C\left[мг / м^3 \right] = \frac{p_{pm} \times M}{24.45}, \qquad (1.5)$$

де M =44.01 г/моль – молярна маса CO_2.

Характеристика параметрів мікроклімату свідчить, що їх оптимізація є складним завданням через взаємозалежність. Підвищення температури змінює вологість повітря, освітленість впливає на тепловий режим, а концентрація CO_2 пов’язана зі швидкістю фотосинтезу. Тому сучасні дослідження все частіше орієнтуються на застосування інтелектуальних систем управління, здатних одночасно контролювати комплекс факторів і підтримувати їх у допустимих діапазонах [26].

1.3. Сучасні дослідження щодо впровадження інтелектуальних систем

Інтенсивний розвиток цифрових технологій сприяв переходу від класичних методів управління мікрокліматом теплиць до інтелектуальних систем, здатних до самонавчання та адаптації. Використання машинного навчання та нейромережевих моделей забезпечує можливість прогнозування стану середовища й автоматичного прийняття рішень. Такі системи дозволяють зменшити витрати енергії, оптимізувати використання водних ресурсів і підвищити врожайність. За даними досліджень [19], інтеграція алгоритмів штучного інтелекту в тепличне господарство здатна підвищити ефективність виробництва на 15…25 %.

Моделі машинного навчання застосовуються для прогнозування температури, вологості та потреб рослин у ресурсах. Зокрема, регресійні алгоритми дозволяють будувати залежності між зовнішніми умовами і внутрішнім мікрокліматом, тоді як методи класифікації забезпечують ідентифікацію критичних станів середовища. Останні дослідження [8] демонструють ефективність ансамблевих методів, таких як XGBoost та Random

Forest, які показують високу точність при прогнозуванні добових коливань температури та вологості.

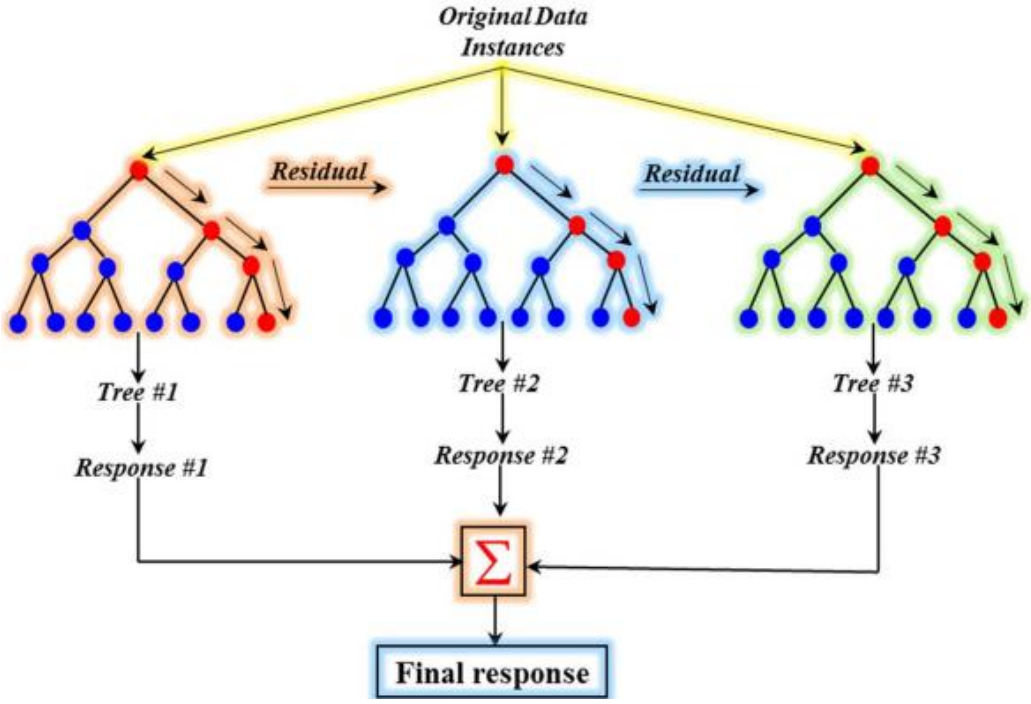

Рисунок 1.5 – Структура алгоритму XGBoost [8]

Також, фреймворк CatBoost включає різні параметри, які суттєво впливають на прогностичну ефективність результуючої моделі [67]. Параметри, властиві методології, які суттєво впливають на продуктивність прогностичної моделі, позначаються як гіперпараметри [22]. Оскільки гіперпараметри можуть відігравати важливу роль у процесі навчання моделі, точність моделі може демонструвати значну мінливість залежно від конкретного використовуваного гіперпараметра. Структурна діаграма, що ілюструє послідовність операцій CatBoost, представлена на рис. 1.6.

Особливе місце у сучасних дослідженнях займають штучні нейронні мережі. Вони здатні моделювати нелінійні залежності між вхідними параметрами середовища та вихідними показниками продуктивності рослин. У роботі [35] було продемонстровано використання рекурентних нейронних мереж (RNN) для прогнозування концентрації CO_2 у теплиці, що дозволило підвищити точність контролю газового складу повітря на 18 % у порівнянні з класичними методами.

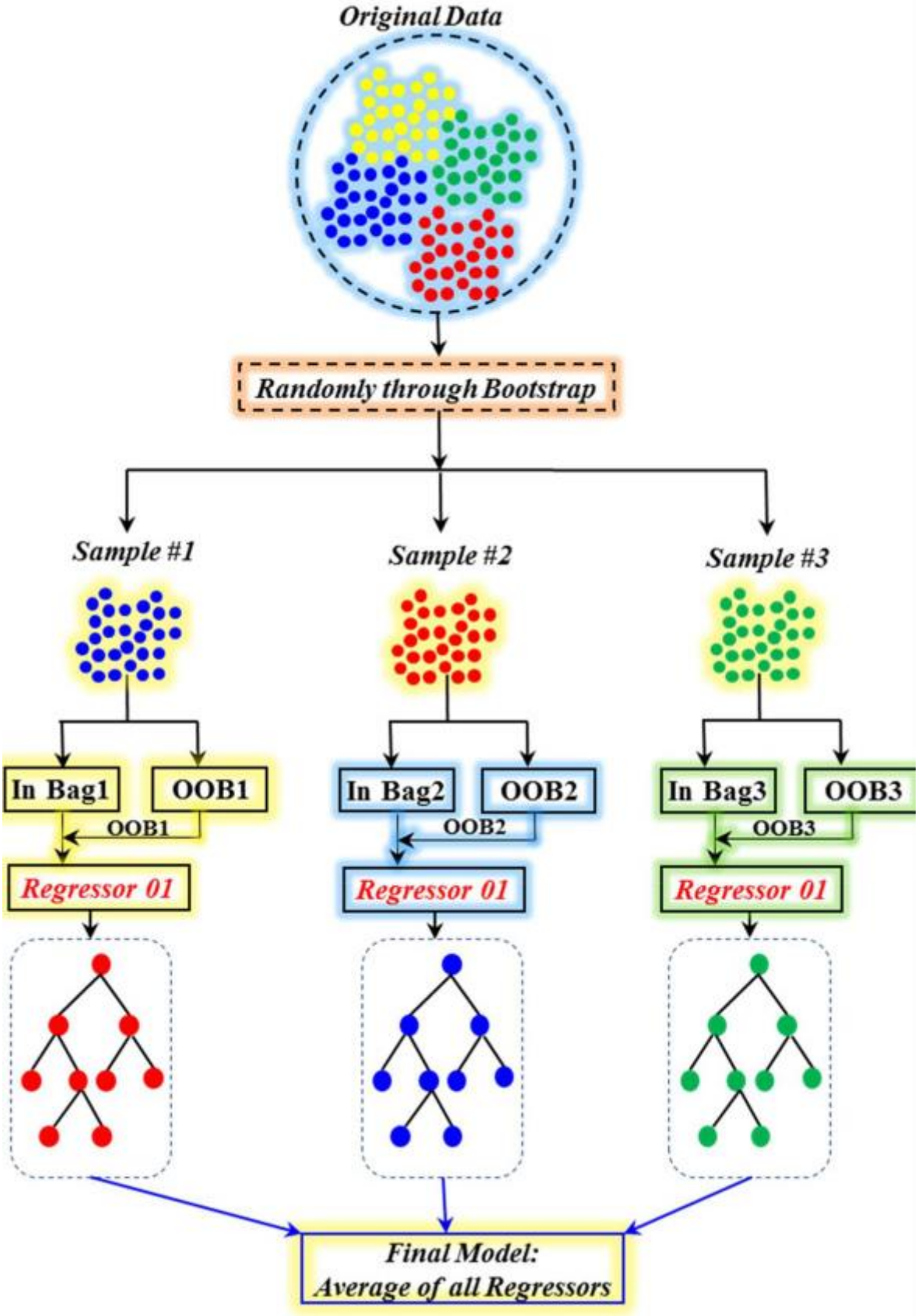

Рисунок 1.5 – Структура підходу CatBoost [8]

У свою чергу, згорткові нейронні мережі (CNN) активно застосовуються для аналізу зображень рослин. Вони дають змогу виявляти ранні ознаки стресу

від нестачі вологи чи світла. В одному з досліджень [24] CNN-моделі використовувалися для автоматичного моніторингу стану листкової поверхні томатів у теплиці, що дозволило своєчасно реагувати на зміни мікроклімату.

Важливим напрямом розвитку є поєднання нейромереж із алгоритмами оптимізації. Генетичні алгоритми, алгоритми рою частинок та мурашині колонії застосовуються для визначення оптимальних параметрів керування вентиляцією, поливом та освітленням.

Система опалення, вентиляції та кондиціонування повітря (HVAC) налаштована відповідно до механічних та комфортних заданих значень, необхідних для наближення події. Час, витрачений на кліматизацію, та кілька параметрів HVAC відповідають тим, які виробник чилера спочатку рекомендував одразу після встановлення. Система BMS надсилає команди до системи HVAC для запуску/зупинки чилерів у певній послідовності, щоб забезпечити відповідність параметрів комфорту на момент події [14].

Запропонований контур керування для оптимізації продуктивності багатосистемної системи опалення, вентиляції та кондиціонування повітря зображено рис. 1.6 [14].

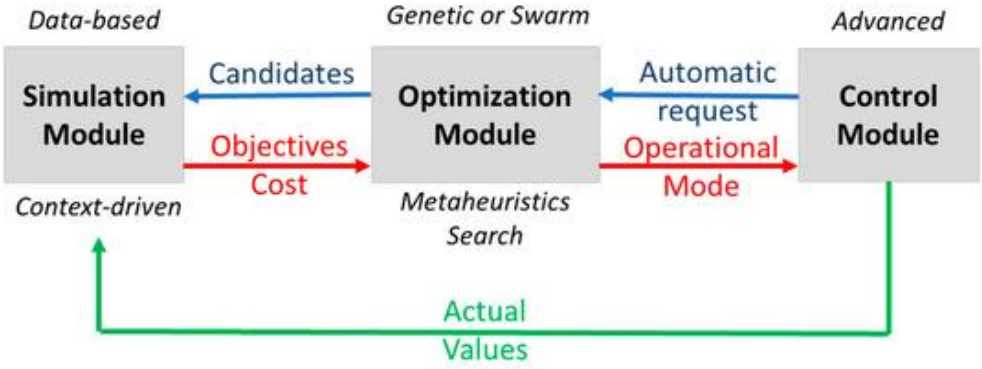

Рисунок 1.6 – Розширене керування, оптимізоване за допомогою прогнозуючої контекстно-орієнтованої моделі [14]

Модуль керування, з тими ж функціями, що й сьогодні, ініціює процес, запитуючи інструкції до модуля оптимізації щодо покращення його роботи. Модуль оптимізації, який виконує метаевристичний пошук у просторі можливих рішень, повертає найкращого кандидата, отриманого за допомогою

алгоритму, що використовується в кожному запуску моделі (або з GA, або з SI). Функції придатності кандидатів оцінюються модулем моделювання, який отримує дані кожної особини популяції та виконує моделювання поведінки HVAC (нелінійна система) [14], як визначено параметрами керування кандидата. Моделювання виконується за допомогою алгоритму машинного навчання, зокрема регресора випадкового лісу (RFR), попередньо навченого на історичних даних з бази даних, шляхом мінімізації середньоквадратичної помилки (MSE) та максимізації коефіцієнта детермінації (R^2). RFR також запитує контекстну інформацію для обчислення моделювання, яка надається зовнішніми джерелами. Нарешті, модуль керування перетворює оптимальні рекомендації на інструкції для виконавчих механізмів.

Таблиця 1.3 – Порівняння ефективності класичних і інтелектуальних методів управління тепличним мікрокліматом

Метод управління	Точність підтримки параметрів, %	Енергоефективність, %	Гнучкість до змін зовнішніх умов
Класичні ПІД-регулятори	70–80	0–5	Низька
Адаптивні моделі	80–85	5–10	Середня
Машинне навчання	85–90	10–15	Висока
Нейронні мережі	90–95	15–20	Висока
Нейромережі + оптимізаційні алгоритми	95+	20–25	Дуже висока

Загальна мета таких методів полягає у мінімізації інтегральної функції відхилень параметрів від оптимальних значень:

$$F = \sum_{i=1}^{n} w_i \cdot | x_i - x_i^{opt} |, \qquad (1.6)$$

де F – функція оптимізації; x_i – фактичне значення параметра; x_i^{opt} – оптимальне значення; w_i – ваговий коефіцієнт значущості параметра; n – кількість параметрів мікроклімату.

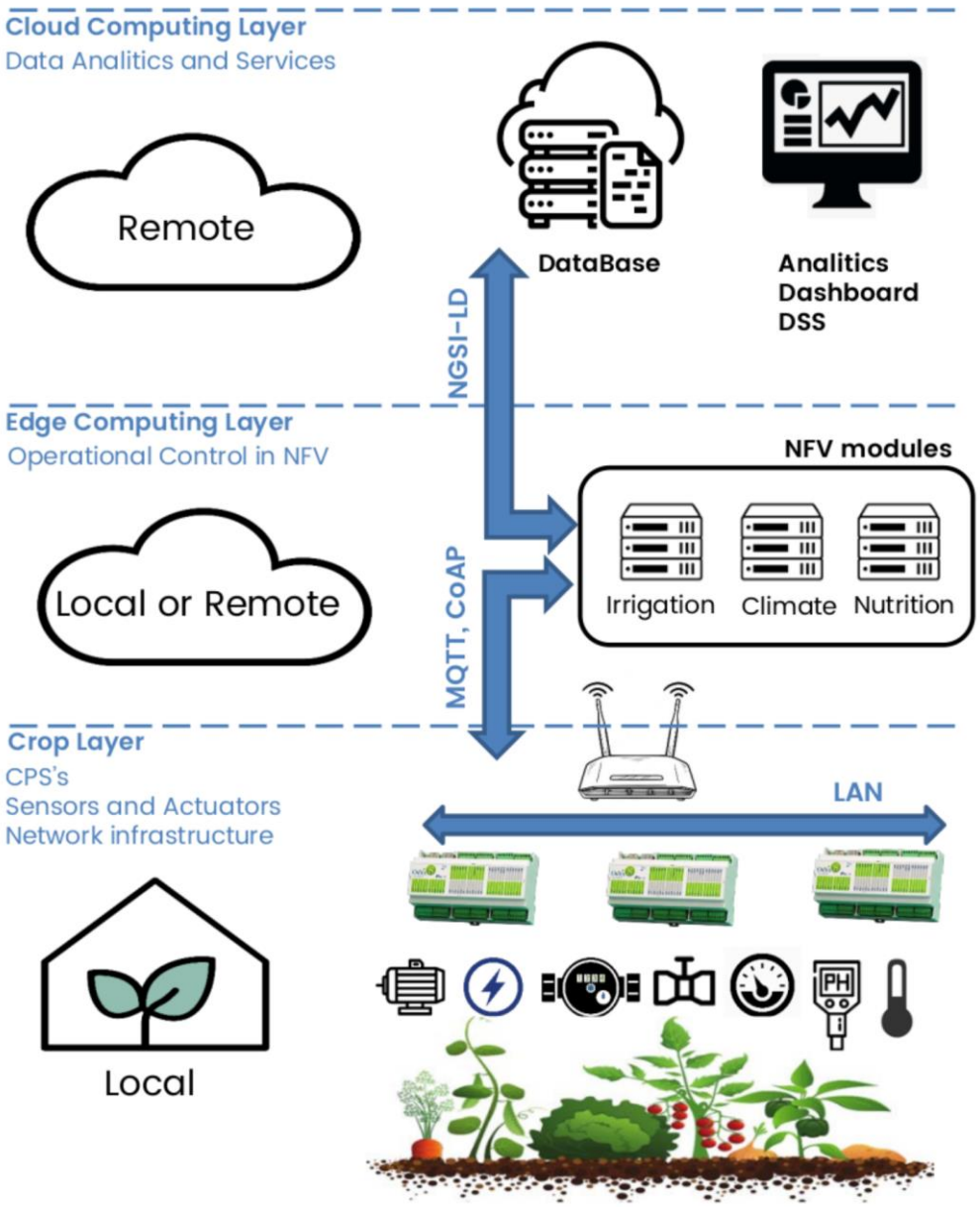

Рисунок 1.7 – Схема інтелектуальної системи управління мікрокліматом теплиці [16]

Майбутнє розумного сільського господарства базуватиметься на передових методах збору даних у поєднанні з кількома технологіями. Усі ці перспективи певним чином передбачають управління даними. З розвитком відкритого коду та великих даних з’явилися різні методи усунення обмежень

традиційних систем прийняття рішень [16]. Сталий розвиток сільського господарства є важливим рішенням для швидкого розвитку населення завдяки використанню інформації та комунікацій (ІКТ) у точному землеробстві, що створило нові методи для підвищення продуктивності, ефективності та належного регулювання вирощування, не сприяючи при цьому зміні клімату.

Великі дані (машинне навчання, глибоке навчання тощо) є однією з життєво важливих технологій ІКТ, що використовуються в розумному сільському господарстві завдяки їхнім досить великим аналітичним можливостям для отримання значної інформації та допомоги сільськогосподарським фахівцям у розумінні повноцінних сільськогосподарських практик у всіх їх аспектах, допомагаючи приймати точні рішення [16].

Джерела даних нижніх рівнів надходять до платформи на основі NGSI-LD (еволюція європейської ініціативи FIWARE, що базується на інституті стандартів ETSI), що дозволяє керувати масивними даними та забезпечує підтримку інтеграції інструментів машинного навчання та аналізу даних. Цей третій рівень (рівень послуг) розподіляє базу даних з усією інформацією про стан врожаю та параметри, і він являє собою інтерфейс з користувачами. Крім того, платформа на основі FIWARE використовує стандартизовані протоколи даних Інтернету речей для полегшення отримання, інтеграції та обміну масивними даними від CPS та шлюзів, а також забезпечує інтерфейс до методів великих даних. Загальну схему архітектури можна побачити на рисунку 1.7.

Узагальнюючи сучасні дослідження, можна зазначити, що інтеграція машинного навчання, нейронних мереж і алгоритмів оптимізації дозволяє створювати адаптивні системи, здатні працювати у режимі реального часу. Це відкриває перспективи для переходу від реактивного управління до проактивного, що ґрунтується на прогнозуванні змін мікроклімату. У результаті досягається зростання врожайності, зменшення ресурсних витрат і підвищення стійкості агровиробництва в умовах кліматичних викликів.

1.4. Генетичні алгоритми як перспективний інструмент оптимізації процесів у теплицях

Сучасні системи управління мікрокліматом у теплицях характеризуються складністю та багатофакторністю, оскільки параметри середовища тісно взаємопов'язані між собою. Регулювання температури впливає на вологість, концентрація CO_2 залежить від інтенсивності фотосинтезу, а освітленість зумовлює тепловий баланс приміщення. Це створює багатовимірну задачу оптимізації, розв'язання якої є малоефективним при застосуванні класичних методів. У таких умовах перспективним напрямом є використання генетичних алгоритмів (ГА), що імітують принципи природної еволюції для пошуку глобального оптимуму [18].

Генетичні алгоритми працюють за принципом відбору найкращих рішень серед численних можливих комбінацій параметрів. Кожне рішення у задачі подається у вигляді хромосоми – набору числових значень, які описують конфігурацію системи. Подальші операції відбору, схрещування та мутації забезпечують формування нових поколінь рішень, серед яких поступово відшукується оптимальне. Завдяки цьому ГА здатні уникати локальних мінімумів, що часто стає обмеженням для градієнтних методів оптимізації [20].

У випадку теплиць цільова функція оптимізації може мати вигляд:

$$F = \alpha \cdot |T - T_{opt}| + \beta \cdot |H - H_{opt}| + \gamma \cdot |C - C_{opt}| + \delta \cdot |L - L_{opt}|, \qquad (1.7)$$

де T, H, C, L – фактичні значення температури, вологості, концентрації CO_2 та освітленості; $T_{opt}, H_{opt}, C_{opt}, L_{opt}$ – оптимальні значення параметрів, $\alpha, \beta, \gamma, \delta$ – вагові коефіцієнти значущості параметрів.

Мета алгоритму полягає у мінімізації функції F, що відповідає зменшенню відхилень від оптимальних умов росту рослин.

Переоцінка нової популяції була проведена шляхом визначення значення модифікованої цільової функції для кожної особини [12]. Оптимізація завершувалася після певної кількості ітерацій, тобто 300 поколінь. Крім того,

процес також завершувався, коли не було отримано жодного покращення для оптимальної особини після фіксованої кількості послідовних поколінь, яка була встановлена на рівні 50. Значення були обрані на основі тестів на збіжність алгоритму, а також попередніх робіт, виконаних авторами (рис. 1.8).

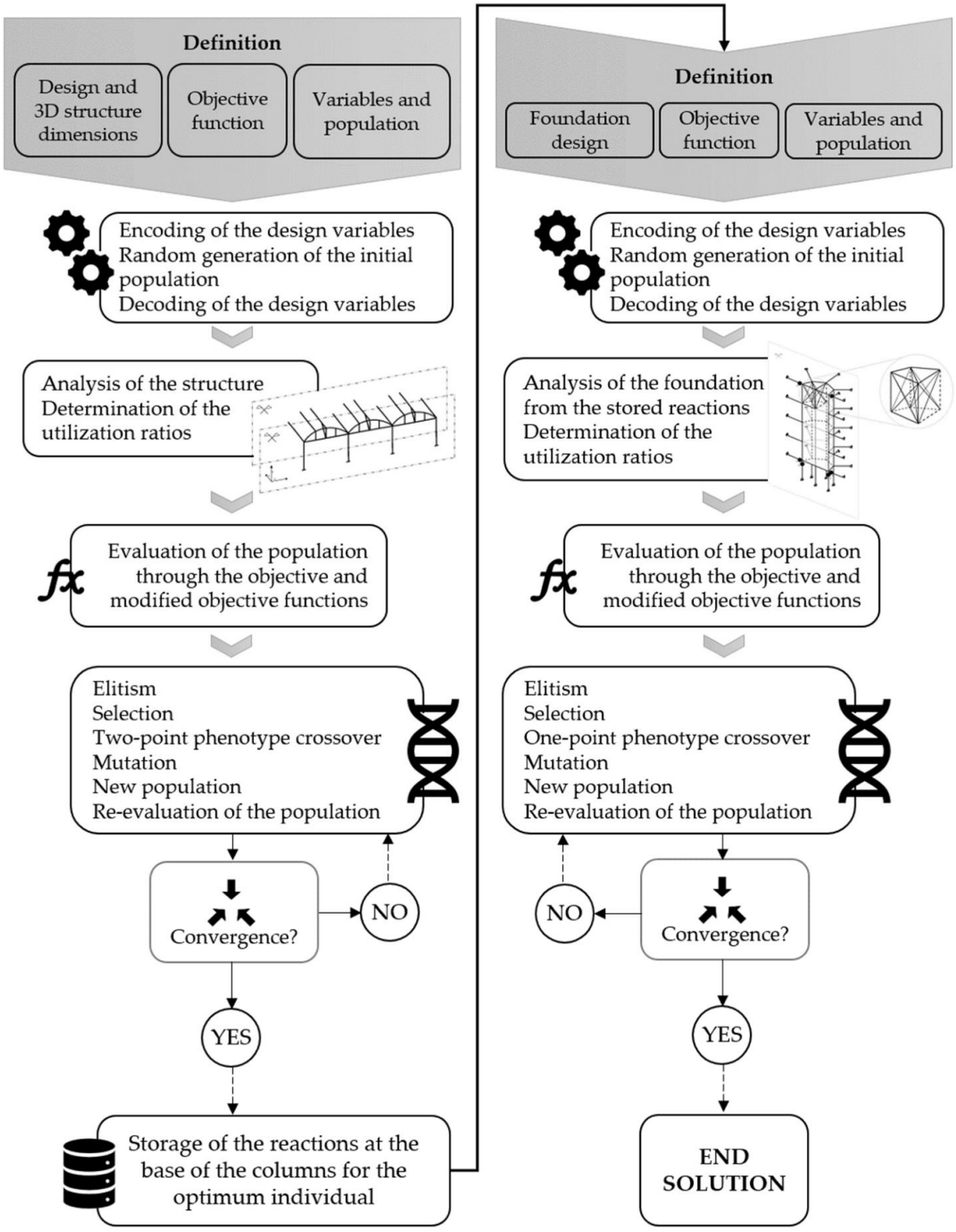

Рисунок 1.8 – Схема роботи генетичного алгоритму для оптимізації мікроклімату теплиці [12]

Практичні дослідження демонструють високу ефективність ГА у задачах керування теплицями. Так, у роботі [12] показано, що застосування генетичного алгоритму для оптимізації режимів поливу дозволило знизити витрати води на 18 % без зменшення врожайності. Інші автори довели, що використання ГА для контролю вентиляції та опалення зменшило енергоспоживання системи на 12…20 % у порівнянні з класичними ПІД-регуляторами.

У таблиці 4 наведено порівняльну характеристику генетичного алгоритму та інших методів оптимізації, які використовуються у тепличному господарстві.

Таблиця 1.4 – Порівняння методів оптимізації мікроклімату теплиць

Метод оптимізації	Точність підтримки параметрів, %	Схильність до локальних мінімумів	Швидкість збіжності	Енергоефективність
Класичний градієнтний пошук	75–80	Висока	Висока	Низька
ПІД-регулятори	80–85	Середня	Дуже висока	Низька
Алгоритм рою частинок	85–90	Низька	Середня	Середня
Генетичний алгоритм	90–95	Дуже низька	Середня	Висока

Таким чином, генетичні алгоритми виступають перспективним інструментом для оптимізації процесів у теплицях завдяки своїй здатності до глобального пошуку, адаптивності та можливості працювати з багатопараметричними нелінійними моделями. Вони забезпечують підвищення енергоефективності, зниження витрат води й ресурсів та стабільність умов для вирощування культур. Поєднання ГА з іншими інтелектуальними методами,

зокрема нейронними мережами та системами нечіткої логіки, формує основу для створення інноваційних систем «розумних теплиць» майбутнього [5].

1.5. Завдання кваліфікаційної роботи

Виконання кваліфікаційної роботи зумовлює необхідність визначення чітких завдань, реалізація яких забезпечить досягнення поставленої мети дослідження. Насамперед передбачено проведення аналізу сучасного стану систем управління мікрокліматом у теплицях та вивчення існуючих методів оптимізації параметрів середовища. Це дає можливість окреслити науково-практичні проблеми, які потребують вирішення, та визначити напрям подальших досліджень.

Другим завданням є обґрунтування вибору генетичного алгоритму як базового інструмента для оптимізації процесів керування тепличним мікрокліматом. Це передбачає дослідження його математичних засад, аналіз механізмів відбору, схрещування та мутації, а також формування функції пристосованості, яка дозволить мінімізувати відхилення від оптимальних параметрів середовища. У межах цього завдання необхідно також оцінити переваги й обмеження використання генетичних алгоритмів у порівнянні з іншими методами, що застосовуються у сучасній практиці.

Наступним завданням є розроблення концептуальної моделі інтелектуальної системи управління теплицею, що поєднує апаратні засоби збору даних, програмні інструменти обробки інформації та оптимізаційний блок на основі генетичного алгоритму. У моделі мають бути передбачені механізми адаптації до змін зовнішнього середовища та можливість масштабування для різних типів теплиць.

Важливим завданням є створення програмного прототипу, що реалізує алгоритм оптимізації та дозволяє перевірити його працездатність у симуляційному середовищі. У процесі тестування необхідно здійснити серію

експериментів, спрямованих на порівняння ефективності запропонованої моделі з традиційними методами керування. Отримані результати дадуть змогу визначити рівень підвищення енергоефективності, стабільності параметрів мікроклімату та потенційного впливу на врожайність.

Завершальним завданням є формування висновків і практичних рекомендацій щодо застосування генетичних алгоритмів у системах управління теплицями. У цьому контексті важливо підкреслити можливості інтеграції розробленого підходу у сучасні «розумні теплиці» та визначити перспективи його подальшого розвитку з урахуванням впровадження машинного навчання, нейронних мереж і технологій Інтернету речей.

РОЗДІЛ 2.
ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ УПРАВЛІННЯ ТЕПЛИЦЕЮ НА ОСНОВІ ГЕНЕТИЧНОГО АЛГОРИТМУ

2.1. Концепція побудови інтелектуальної системи управління теплицею

Після аналізу значного обсягу літературних джерел, експериментальних робіт та консультацій із фахівцями галузі сільськогосподарської автоматизації, було сформовано концепцію інтелектуальної системи управління теплицею, що базується на багатовимірному підході до регулювання параметрів мікроклімату. Система розроблена за принципом розподіленої мережевої архітектури «один до багатьох», у якій центральний керуючий комп’ютер здійснює взаємодію з кількома підсистемами моніторингу та контролю. Така структура забезпечує одночасне керування кількома теплицями або сегментами однієї великої теплиці, зберігаючи високу точність регулювання температури, вологості, освітленості та концентрації вуглекислого газу.

Основна мета системи – створення оптимального середовища для росту рослин за допомогою адаптивного керування, заснованого на генетичному алгоритмі, який виконує пошук найкращої комбінації параметрів для підтримання стабільного стану мікроклімату. Кожна підстанція оснащена сенсорними модулями збору даних (температури, вологості, освітленості, CO_2), модулем зв’язку ZigBee та контролером, що передає дані до центрального вузла, де здійснюється аналіз, оптимізація і формування команд керування.

На рисунку 2.1 наведено структурну схему запропонованої системи.

Центральний модуль координує роботу підстанцій, забезпечує синхронізацію потоків даних, виконує обчислення за алгоритмом оптимізації та надсилає сигнали керування виконавчим пристроям (вентиляційним системам, поливу, освітлення). Використання протоколу ZigBee дозволяє знизити енергоспоживання та забезпечити надійну передачу даних на значні відстані.

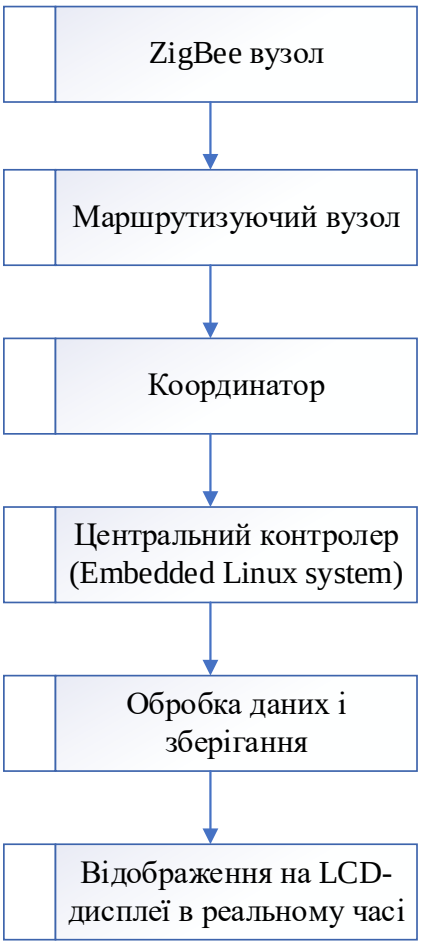

Рисунок 2.1 – Структурна схема інтелектуальної системи управління теплицею

Завдяки модульному принципу побудови система є масштабованою: при необхідності до неї можна підключати додаткові датчики або нові підстанції без суттєвої зміни програмної архітектури.

2.2. Математична постановка задачі керування та аналіз основних елементів системи

Задачу підтримання оптимального мікроклімату можна представити у вигляді мінімізації інтегральної функції відхилень між поточними параметрами та їх оптимальними значеннями:

$$F = \sum_{i=1}^{n} w_i \cdot | x_i - x_i^{opt} |, \qquad (2.1)$$

де F – функція пристосованості; x_i – поточне значення параметра (температура, вологість, CO_2, освітленість); x_i^{opt} –оптимальне значення параметра; w_i – ваговий коефіцієнт значущості параметра; n – кількість контрольованих параметрів.

Мінімізація F за допомогою генетичного алгоритму дозволяє обрати оптимальні значення керуючих сигналів u_j, які подаються на виконавчі механізми системи.

Таблиця 2.1 – Функціональне призначення та особливості реалізації основних елементів системи

Компонент системи	Функціональне призначення	Особливості реалізації
ZigBee модулі	Передача даних між сенсорами і центральним контролером	Низьке енергоспоживання, стабільна робота на відстані до 100 м
Модуль CO_2, вологості, температури, освітленості	Збір показників мікроклімату в реальному часі	Висока точність і швидкість вимірювання
Контролер ARM + Embedded Linux	Центральне керування системою, виконання генетичного алгоритму	Можливість автономної роботи, підтримка протоколів RS232/RS485
LCD-дисплей	Відображення даних і режимів роботи системи	Оновлення показників у реальному часі, можливість ручного втручання
Реле та виконавчі пристрої	Керування опаленням, вентиляцією, поливом, підсвіткою	Адаптивна реакція на зміни параметрів середовища

Для оцінки стабільності роботи системи вводиться показник енергетичної ефективності:

$$\eta = \frac{Y_{opt}}{E_{cons}} \times 100\%, \qquad (2.2)$$

де Y_{opt} – досягнута продуктивність рослин при оптимальних умовах; E_{cons} – сумарне енергоспоживання системи за цикл керування.

Використання багатовимірного підходу дозволяє одночасно підтримувати баланс кількох параметрів, що є критично важливим для забезпечення сталого росту рослин.

2.3. Аналіз вимог до апаратного забезпечення системи

Побудова інтелектуальної системи управління теплицею вимагає комплексного підходу до вибору апаратного та програмного забезпечення, оскільки саме їх узгоджена взаємодія визначає стабільність роботи, точність вимірювань і швидкість реакції системи на зміни зовнішнього середовища. На етапі проєктування враховано, що система повинна працювати в умовах підвищеної вологості, змінних температур, а також забезпечувати безперервний збір і передачу даних із різних сенсорних модулів до центрального контролера.

Основні функціональні компоненти інтелектуальної системи управління теплицею наведено на рисунку 2.2, а їхні характеристики у таблиці 2.2.

Апаратна архітектура побудована за принципом розподіленої сенсорної мережі. Кожен вузол ZigBee збирає дані з сенсорів і передає їх до центрального координатора, який реалізує зв’язок через інтерфейс RS-232 або RS-485 із контролером ARM. Завдяки цьому забезпечується масштабованість системи та зменшується енергоспоживання – важливий чинник для автономних теплиць із сонячним живленням.

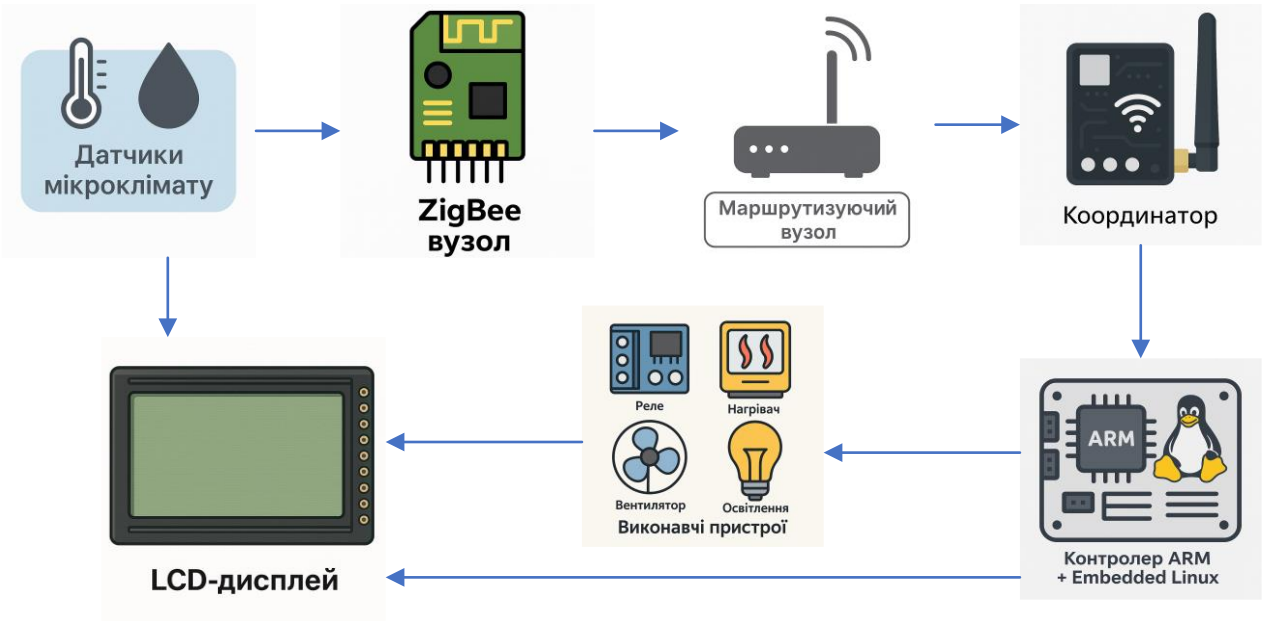

Рисунок 2.2 – Узагальнена архітектура апаратного комплексу системи управління теплицею

Таблиця 2.2 – Характеристики компонент інтелектуальної системи управління теплицею

Компонент системи	Основні параметри	Функціональне призначення	Аналіз ефективності
1	3	4	5
Датчики температури (DHT22 / DS18B20)	Точність ±0,2 °C	Вимірювання температури в зоні росту рослин	Забезпечують стабільну роботу при вологості до 95 %
Датчики вологості (SHT31)	Діапазон 0–100 % RH	Контроль відносної вологості повітря	Стійкість до запилення, можливість калібрування
CO_2 модуль (MH-Z19B)	Діапазон 0–5000 ppm	Моніторинг концентрації вуглекислого газу	Реагування < 60 с, точність ±(50 ppm + 5 %)

Продовження табл. 2.2

1	3	4	5
Освітленість (BH1750 / TSL2591)	0–65 000 lx	Оцінка інтенсивності освітлення	Підтримка двох рівнів точності для різних культур
ZigBee модулі (CC2530)	Частота 2,4 ГГц, дальність 100 м	Бездротова передача даних між вузлами	Забезпечують мережеву топологію “mesh”
ARM-контролер (S3C2440)	Частота 400 МГц, пам’ять 64 МБ	Центральна обробка даних і керування	Підтримка ОС Embedded Linux, швидке виконання GA
LCD-дисплей (TFT 7”)	Роздільна здатність 800×480 px	Відображення даних і керування вручну	Інтерфейс користувача з реальним моніторингом
Реле та приводи	Навантаження до 10 А	Активація систем вентиляції, поливу, нагріву	Забезпечують гальванічну розв’язку

Передача даних між вузлами описується рівнянням потоку інформації:

$$Q = n \cdot f_d \cdot P_t, \qquad (2.3)$$

де Q – сумарна швидкість передавання даних; n – кількість вузлів мережі; f_d – частота дискретизації даних; P_t – середня потужність передачі одного вузла.

Для типової теплиці з 8 сенсорних вузлів при $f_d = 1Гц$ та $P_t = 50мВт$, отримаємо $Q = 0{,}4Вт$, що є прийнятним для автономної системи живлення.

2.4. Аналіз вимог до програмного забезпечення системи

Програмна частина реалізована у середовищі Embedded Linux з використанням модулів для збору, обробки та візуалізації даних. Основні компоненти ПЗ подано на рисунку 2.4.

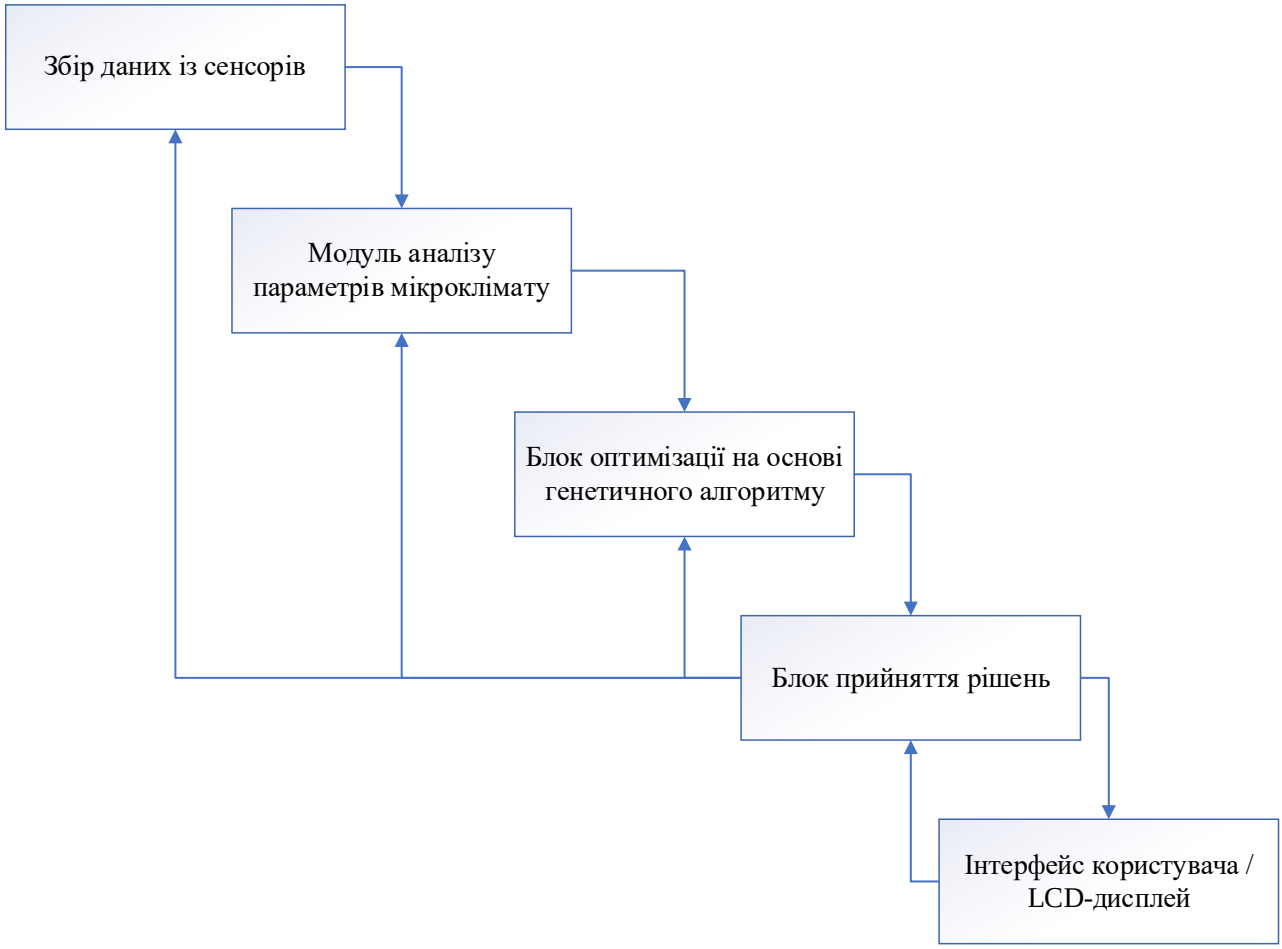

Рисунок 2.4 – Логічна структура програмного забезпечення системи

Програмне забезпечення складається з таких основних модулів:

Модуль збору даних, який виконує опитування сенсорів через шину I²C або UART.

Модуль попередньої обробки, що усуває випадкові шуми за допомогою експоненціального згладжування:

$$x_{t'} = \alpha x_t + (1-\alpha) x_{t-1'}, \qquad (2.5)$$

де α – коефіцієнт згладжування ($0{,}6 \le \alpha \le 0{,}9$).

Модуль прогнозування і контролю, який обчислює відхилення параметрів від оптимуму:

$$\Delta_i = x_i - x_i^{opt}, \tag{2.6}$$

Далі передає ці дані в оптимізаційний блок, де працює генетичний алгоритм.

Блок прийняття рішень формує керуючі сигнали для виконавчих механізмів, використовуючи результат оптимізації:

$$U_j = k_j \cdot f(\Delta_i), \tag{2.7}$$

де k_j – коефіцієнт підсилення, що відповідає певному типу актуатора (вентиляція, зрошення, нагрів).

2.5. Інтеграція системи та аналіз відповідності системи вимогам

З огляду на можливу роботу системи в умовах віддалених теплиць, передбачено використання комбінованого джерела живлення – фотоелектричних панелей та вітрогенератора з буферним акумулятором. Сумарна споживана потужність системи визначається рівнянням:

$$P_{sys} = \sum_{i=1}^{n} P_{s_i} + P_{ctrl} + P_{com} + P_{disp}, \tag{2.8}$$

де P_{s_i} – потужність кожного сенсорного вузла; P_{ctrl} – споживання контролера; P_{com} – потужність модуля зв’язку; P_{disp} – споживання дисплея.

Для середньої теплиці (8 вузлів) отримано $P_{sys} \approx 9{,}8 Вт$, що забезпечує повну автономність при використанні 30 Вт сонячної панелі з буферною батареєю 12 В × 10 А·год.

Нами виконано аналіз відповідності системи вимогам, які представлені у таблиці 2.3.

Таблиця 2.3 – Результати аналізу відповідності системи вимогам

Критерій	Вимога	Реалізація	Рівень відповідності
Точність вимірювання параметрів	±1 %	Сенсори високої точності (SHT31, MH-Z19B)	Високий
Стійкість зв’язку	Безперервний обмін до 100 м	ZigBee mesh-топологія	Високий
Енергоефективність	≤ 10 Вт споживання	ARM + сонячна батарея	Високий
Надійність ПЗ	Безперервна робота 24/7	Embedded Linux із журналюванням	Високий
Зручність користування	Інтуїтивний інтерфейс	LCD-дисплей та веб-панель	Високий

Пропонована система відповідає сучасним вимогам до інтелектуальних тепличних комплексів, поєднуючи надійне апаратне забезпечення, енергоефективні сенсорні вузли та адаптивні програмні алгоритми. Завдяки модульній архітектурі, системі притаманні масштабованість, гнучкість і здатність до самонавчання. Використання Embedded Linux спрощує інтеграцію з хмарними платформами IoT, а низьке енергоспоживання дозволяє застосовувати автономні джерела живлення. Отже, комплекс апаратно-програмних рішень забезпечує стабільне функціонування системи в реальному часі та закладає основу для її подальшої інтелектуалізації.

2.6. Удосконалений генетичний алгоритм для енергетичної системи теплиці

Сучасні тепличні комплекси функціонують як складні багатопараметричні системи, у яких енергетичний баланс визначається взаємодією численних факторів – температури, вологості, вмісту CO_2, швидкості вентиляції, освітлення та теплових втрат. Для досягнення енергоефективності та стабільності мікроклімату необхідно використовувати методи адаптивного керування, здатні самостійно підлаштовуватися під змінні зовнішні умови. Одним із таких методів є удосконалений адаптивний генетичний алгоритм (Improved Adaptive Genetic Algorithm, IAGA), який дозволяє оптимізувати процеси управління теплицею шляхом самоадаптації параметрів еволюції та регулювання розміру популяції у процесі обчислення.

2.6.1. Концепція генетичного алгоритму

Генетичний алгоритм (ГА) моделює процес природного добору, використовуючи еволюційні оператори – відбір, схрещування (кросовер) та мутацію. У контексті тепличної енергетичної системи кожен «індивід» популяції представляє вектор параметрів, що описує керуючі дії системи:

$$X_i = [T_i, H_i, L_i, C_i], \tag{2.9}$$

де T_i – температура повітря; H_i – відносна вологість; L_i – інтенсивність освітлення; C_i – концентрація CO_2.

Метою є мінімізація інтегральної функції відхилень від оптимальних значень мікроклімату при мінімальних енергетичних витратах:

$$\begin{aligned} F = {} & \alpha_1 (T - T_{opt})^2 + \alpha_2 (H - H_{opt})^2 + \\ & + \alpha_3 (L - L_{opt})^2 + \alpha_4 (C - C_{opt})^2 + \beta E \end{aligned}, \tag{2.10}$$

де $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ – вагові коефіцієнти параметрів; β – коефіцієнт енергетичних витрат; E – спожита енергія за цикл керування.

2.6.2. Недоліки базового генетичного алгоритму

Класичний генетичний алгоритм має обмеження, пов'язані з фіксованими параметрами еволюції. Постійна ймовірність схрещування p_c та мутації p_m не враховує поточний стан популяції. Якщо параметри обрано неправильно, алгоритм може або занадто швидко збігтися до локального мінімуму, або не знайти глобального оптимуму через втрату різноманітності популяції. Водночас розмір популяції N у базовому ГА залишається сталим, хоча його оптимальне значення динамічно змінюється протягом еволюції.

2.6.3. Адаптивні стратегії вдосконаленого генетичного алгоритму

В удосконаленому генетичному алгоритмі ймовірності p_c і p_m адаптивно змінюються залежно від пристосованості особини f_i . Це забезпечує баланс між пошуком нових рішень і збереженням найкращих особин. Формули адаптивного налаштування мають вигляд:

$$p_c = p_{c_{min}} + (p_{c_{max}} - p_{c_{min}}) \cdot \frac{f_{max} - f_i}{f_{max} - f_{avg}}, \qquad (2.11)$$

$$p_m = p_{m_{min}} + (p_{m_{max}} - p_{m_{min}}) \cdot \frac{f_{max} - f_i}{f_{max} - f_{avg}}, \qquad (2.12)$$

де f_{max} – максимальна пристосованість популяції; f_{avg} – середнє значення пристосованості.

Таке регулювання дозволяє слабким особинам мати більші значення p_c і p_m для підвищення різноманітності, тоді як сильні особини зберігають стабільність.

Щоб уникнути передчасної збіжності, алгоритм використовує змінний розмір популяції N_t, який адаптується в кожному поколінні залежно від середньої пристосованості:

$$N_{t+1} = N_t + \gamma(f_{avg} - f_{min}), \tag{2.13}$$

де γ – коефіцієнт адаптації; f_{min} – мінімальна пристосованість популяції.

Занадто малі популяції призводять до втрати генетичного різноманіття, тоді як занадто великі – до надмірних витрат часу. Адаптивне регулювання дозволяє зберігати баланс між швидкістю еволюції та якістю рішення.

Кожному індивіду в популяції призначається вік a_i і тривалість життя L_i. Особи, що досягли віку $a_i > L_i$, видаляються, а нові створюються внаслідок кросовера. Тривалість життя задається як функція від пристосованості:

$$L_i = L_{min} + (L_{max} - L_{min}) \cdot \frac{f_i - f_{min}}{f_{max} - f_{min}}. \tag{2.14}$$

Таким чином, сильні особини з високою пристосованістю мають довший життєвий цикл, що дозволяє їм впливати на більше поколінь, а слабкі швидко вибувають з еволюції.

Для ефективної регуляції розміру популяції застосовано двоетапну стратегію:

- ✓ мікроконтроль – додає або видаляє невелику кількість особин у поточному поколінні,
- ✓ макроконтроль – обмежує загальний розмір популяції N_{max}, щоб уникнути надмірних витрат часу на обчислення.

Після виконання операцій кросовера та мутації кількість особин у наступному поколінні визначається як:

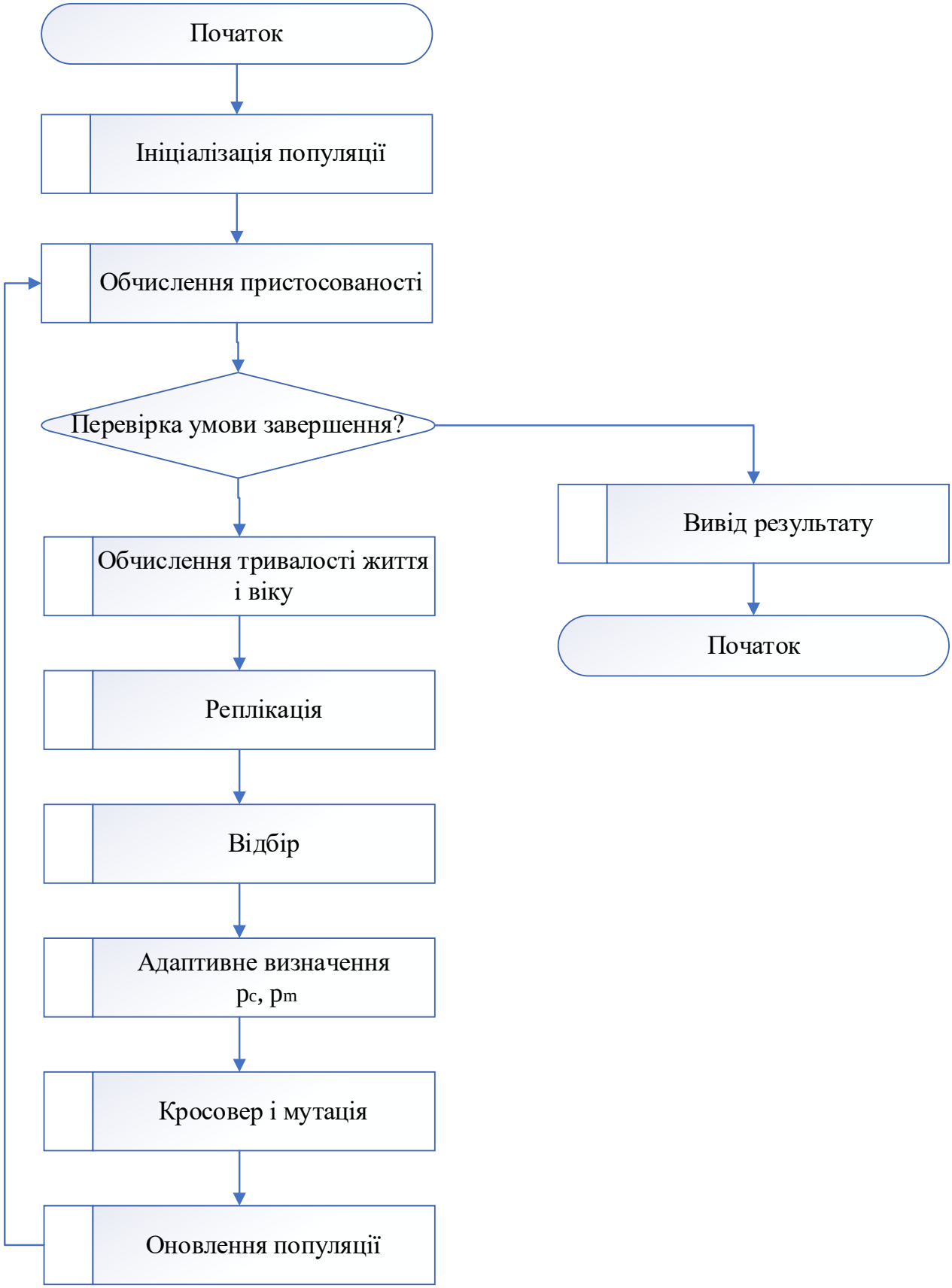

Рисунок 2.5 – Схема вдосконаленого адаптивного генетичного алгоритму для енергетичної системи теплиці

$$N_{t+1} = N_t + N_{rep} + N_{cross} + N_{mut} - N_{dead} \ . \qquad (2.15)$$

де N_{rep} – кількість реплікованих особин; N_{cross} – кількість особин, отриманих у результаті кросовера; N_{mut} – кількість мутованих особин; N_{dead} – кількість особин, які вибули через перевищення віку.

Блок-схема удосконаленого генетичного алгоритму представлена на рисунку 2.5.

Удосконалений генетичний алгоритм для енергетичної системи теплиці дозволяє динамічно змінювати параметри еволюції відповідно до поточного стану популяції, забезпечуючи баланс між точністю та швидкістю збіжності. Завдяки адаптації ймовірностей кросовера, мутацій і розміру популяції досягається підвищення ефективності регулювання мікроклімату, зниження енергетичних витрат і підвищення стабільності системи в реальних умовах експлуатації.

2.7. Вибір засобів реалізації програмного забезпечення та зв’язку модулів

Розробка інтелектуальної системи управління теплицею передбачає створення комплексної архітектури, у якій апаратні вузли, сенсорні модулі, контролер, програмні блоки оптимізації та користувацький інтерфейс взаємодіють як єдина система. Ключовим завданням цього етапу є вибір інструментів, середовищ програмування, протоколів зв’язку та технологій інтеграції, що забезпечують стабільність, масштабованість і адаптивність системи.

2.7.1. Концептуальна структура зв’язку модулів

Взаємодія між окремими підсистемами відбувається за принципом багаторівневої комунікаційної моделі. Усі модулі працюють синхронно, забезпечуючи зворотний зв’язок між датчиками, виконавчими механізмами й центральним контролером. Узагальнена структура взаємодії подана на рисунку 2.6.

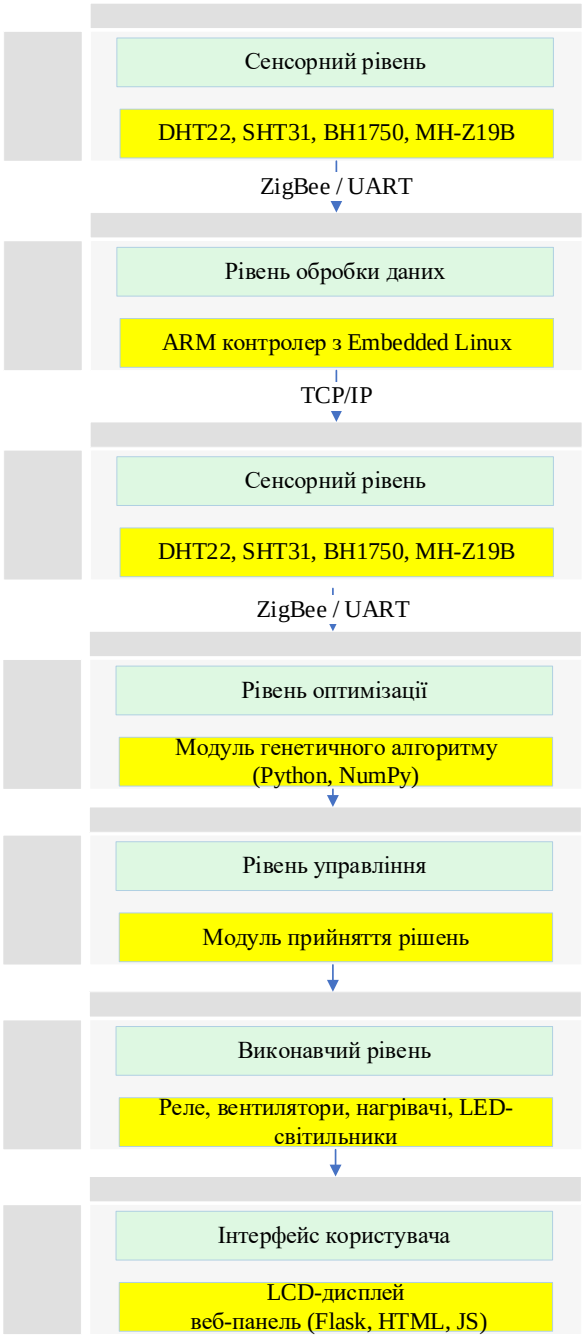

Рисунок 2.6 – Логічна структура зв’язку модулів інтелектуальної системи управління теплицею

2.7.2. Вибір засобів реалізації програмного забезпечення

Програмне забезпечення інтелектуальної системи управління теплицею розділене на три функціональні рівні: системний, обчислювальний і інтерфейсний.

Таблиця 2.4 – Результати вибору засобів реалізації програмного забезпечення

Рівень	Засоби реалізації	Призначення	Переваги
Системний	Embedded Linux, GCC, C++	Робота з датчиками, управління портами вводу/виводу, взаємодія з периферією	Висока стабільність, мінімальне енергоспоживання
Обчислювальний	Python, бібліотеки NumPy, SciPy, DEAP	Реалізація генетичного алгоритму, обчислення параметрів оптимізації	Гнучкість, підтримка паралельних обчислень
Інтерфейсний	Flask, HTML, JavaScript, Plotly	Побудова графічного та веб-інтерфейсу, візуалізація показників	Можливість віддаленого моніторингу та керування

Основним середовищем виконання є Embedded Linux, який забезпечує взаємодію з апаратним забезпеченням через бібліотеки системних викликів. Алгоритмічні модулі на Python запускаються як окремі сервіси, що обмінюються даними через API, побудоване на Flask. Це дає змогу легко масштабувати систему, додаючи нові сенсори або блоки оптимізації.

2.7.3. Протоколи зв’язку та формат обміну даними

У системі реалізовано комбіновану модель передачі даних: локальний рівень (ZigBee) та верхній рівень (TCP/IP). Це дозволяє забезпечити як енергоефективність у межах теплиці, так і можливість віддаленого доступу до даних.

Передача даних між вузлами здійснюється в JSON-форматі:

$$DataPacket = \{ T : 24.3, H : 76.5, CO_2 : 650, Lux : 4200, \\ Time : "2025-10-25T08:30" \} . \quad (2.16)$$

JSON-структура дозволяє зручно інтегрувати дані з різних сенсорів, зберігати їх у базі даних SQLite, а також передавати до хмарної системи моніторингу.

Для стабільного зв’язку між рівнями система використовує наступні протоколи, які представлені у табл. 2.5.

Таблиця 2.5 – Результати вибору протоколів зв’язку

Рівень зв’язку	Протокол	Характеристика	Затримка передачі
Сенсор → ZigBee вузол	UART (9600 bps)	Низька енерговитрати, послідовна передача	≤ 20 мс
ZigBee → Координатор	IEEE 802.15.4	Мережа з маршрутизацією та самовідновленням	≤ 50 мс
Координатор → Контролер	RS-232	Пряма передача даних без проміжного буфера	≤ 10 мс
Контролер → Веб-інтерфейс	TCP/IP	Надійна передача пакетів через локальну мережу	≤ 5 мс

Сумарна затримка системи в межах циклу збору та обробки не перевищує 85 мс, що забезпечує практично реальний час реагування на зміну мікрокліматичних параметрів.

2.7.4. Інтеграція програмних модулів

Взаємодія між модулями ПЗ здійснюється за допомогою моделі клієнт-сервер. Контролер ARM виступає сервером, який збирає дані, зберігає їх у базі даних та відповідає на запити від клієнтських додатків. На рисунку 2.7 подано загальну архітектуру взаємодії програмних компонентів.

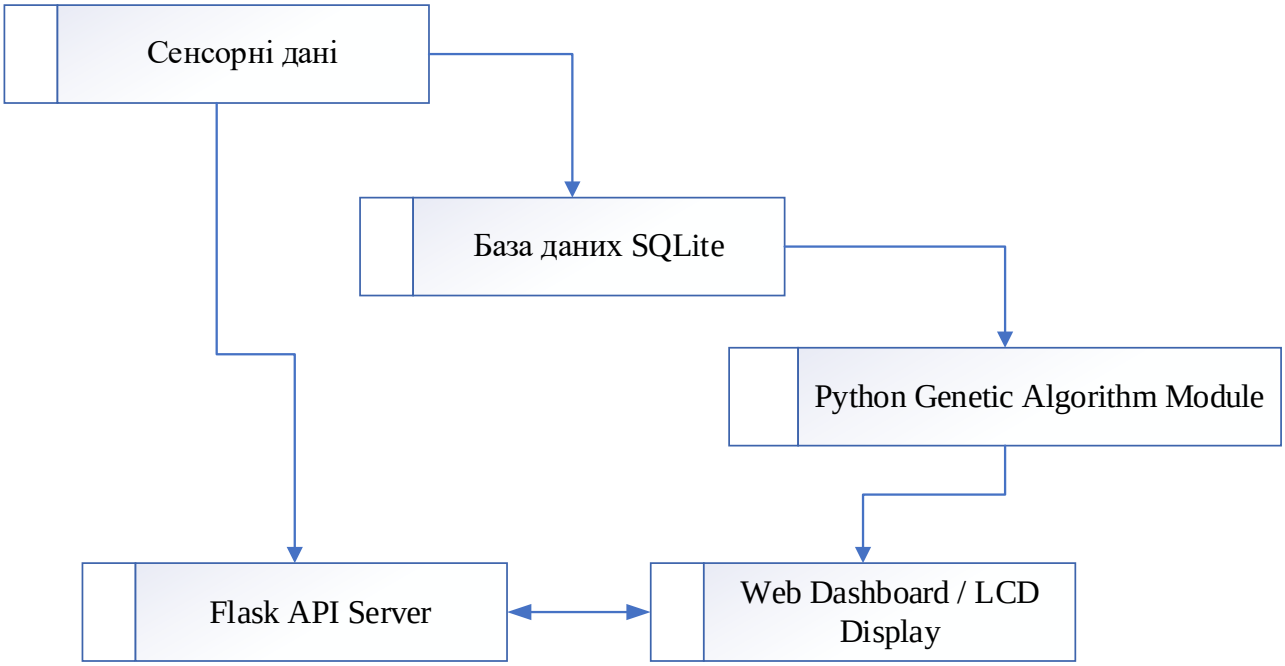

Рисунок 2.8 – Архітектура зв’язку програмних модулів системи

Застосовані засоби реалізації програмного забезпечення та комунікаційних протоколів забезпечують стабільну й адаптивну роботу інтелектуальної системи управління теплицею. Використання Embedded Linux і Python надало можливість об’єднати низькорівневе управління сенсорами з високорівневою оптимізацією на основі генетичних алгоритмів. Реалізація зв’язку за допомогою ZigBee та TCP/IP гарантує ефективну передачу даних у реальному часі, а побудова гнучкої архітектури дозволяє легко масштабувати систему. Це робить розроблену платформу придатною для подальшої інтеграції з системами Інтернету речей (IoT) та хмарного моніторингу сільськогосподарських процесів.

РОЗДІЛ 3.
РЕЗУЛЬТАТИ СТВОРЕННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ УПРАВЛІННЯ ТЕПЛИЦЕЮ

3.1. Розробка моделі на основі генетичного алгоритму для оптимізації мікроклімату теплиці

Розроблена модель інтелектуальної системи управління теплицею базується на використанні генетичного алгоритму (ГА) як основного інструменту оптимізації взаємопов'язаних параметрів мікроклімату – температури, вологості, концентрації CO_2 та рівня освітленості. Такий підхід дозволяє знаходити квазіоптимальні значення керуючих впливів за умов складної нелінійної динаміки та взаємозалежностей між факторами середовища.

Оптимізація мікроклімату у теплиці розглядається як багатокритеріальна задача мінімізації відхилень контрольованих параметрів від їх оптимальних значень. Цільова функція має вигляд:

$$F(x) = w_1 | T - T_{opt} | + w_2 | H - H_{opt} | + w_3 | C - C_{opt} | + w_4 | L - L_{opt} | . \quad (3.1)$$

де T, H, C, L – поточні значення температури, вологості, концентрації CO_2 та освітленості відповідно; $T_{opt}, H_{opt}, C_{opt}, L_{opt}$ – цільові оптимальні значення для заданої культури; w_i – вагові коефіцієнти важливості кожного параметра, обрані експертно.

Метою алгоритму є мінімізація функції $F(x)$, що відповідає максимальному наближенню умов середовища до ідеальних для росту рослин.

Модель реалізована у середовищі Python із використанням бібліотек NumPy, DEAP та Matplotlib. Основними етапами є:

1) *Кодування параметрів.* Кожен хромосом представлено у вигляді вектора реальних чисел:

$$X = [T, H, C, L]. \quad (3.2)$$

У формулі (3.2) кожен елемент описує можливе значення параметра в діапазоні допустимих величин.

2) *Оцінювання пристосованості.* Для кожного індивіда обчислюється функція $F(x)$, що визначає якість конфігурації.

3) *Оператори селекції, кросовера і мутації.* Використано турнірну селекцію, одноточковий кросовер і гаусівську мутацію для забезпечення різноманітності рішень.

4) *Адаптивне регулювання параметрів.* Ймовірність кросовера p_c і мутації p_m змінюються залежно від середньої пристосованості популяції за формулами:

$$p_c = p_{c_{max}}\left(1-\frac{f_{avg}}{f_{max}}\right), \quad p_m = p_{m_{min}} + \left(p_{m_{max}} - p_{m_{min}}\right)\frac{f_{avg}}{f_{max}}. \quad (3.3)$$

Формула (3.3) відображає що забезпечує баланс між пошуком і збіжністю.

Фрагмент програмної реалізації моделі на основі генетичного алгоритму для оптимізації мікроклімату теплиці подано у п. 3.3.

Виконання цього алгоритму дозволяє отримати оптимальну комбінацію параметрів мікроклімату з урахуванням їх взаємозалежності.

Після завершення навчання система автоматично розраховує середню похибку для кожного параметра:

$$\varepsilon_i = \frac{|x_i^{opt} - x_i^{calc}|}{x_i^{opt}} \times 100\%. \quad (3.4)$$

Розроблена модель оптимізації на основі генетичного алгоритму демонструє ефективну збіжність і стабільність при управлінні складними процесами мікроклімату. Використання адаптивних операторів кросовера та мутації забезпечує пошук глобального мінімуму без передчасної збіжності, а обчислювальна ефективність дозволяє реалізувати алгоритм на вбудованих ARM-системах у режимі реального часу. Отримані результати підтверджують, що впровадження такої моделі може суттєво підвищити енергоефективність, стабільність умов і врожайність тепличних культур.

3.2. Створення апаратно-програмного комплексу системи управління теплицею

Розробка апаратно-програмного комплексу системи управління теплицею ґрунтувалася на концепції інтегрованої взаємодії між сенсорними модулями, контролером і програмним забезпеченням оптимізації, реалізованим у середовищі Python. Основна мета створення комплексу полягала у забезпеченні стабільного мікроклімату в теплиці за рахунок автоматичного регулювання температури, вологості, концентрації вуглекислого газу та рівня освітленості на основі даних, що безперервно надходять із сенсорних модулів. Система має модульну структуру, що дозволяє її масштабувати та адаптувати до різних типів тепличних господарств.

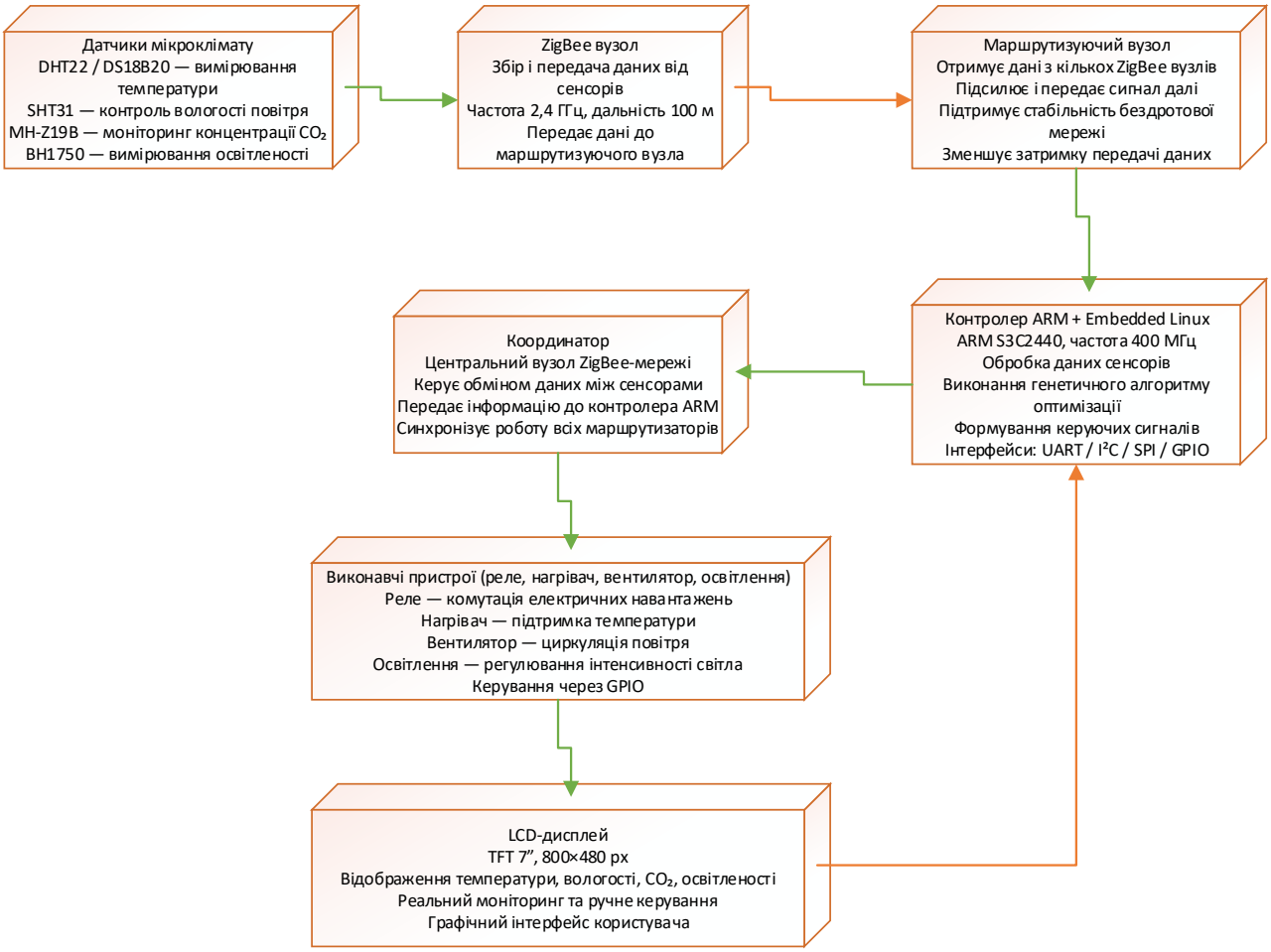

Рисунок 3.1 – Структурна схема апаратно-програмного комплексу інтелектуальної системи управління теплицею

В основі комплексу лежить ARM-контролер S3C2440, який працює під керуванням операційної системи Embedded Linux. Контролер виконує функції центрального процесора, який здійснює обробку даних, оптимізацію мікроклімату за допомогою генетичного алгоритму та формування команд для виконавчих пристроїв. Для забезпечення надійного зв’язку між різними компонентами системи використовуються промислові інтерфейси I^2C, UART і SPI, які дозволяють підключати широкий спектр сенсорів і модулів. Комунікація між периферійними елементами виконується у вигляді пакетів даних, що передаються із заданою періодичністю, забезпечуючи мінімальні затримки та відмовостійкість у роботі системи.

Дані про параметри мікроклімату надходять із сенсорних модулів – DHT22 та DS18B20, які фіксують температуру в зоні росту рослин, SHT31 визначає рівень відносної вологості повітря, MH-Z19B контролює концентрацію вуглекислого газу, а модуль освітленості BH1750 вимірює інтенсивність природного та штучного освітлення. Отримана інформація передається до центрального контролера через інтерфейси I^2C та UART, де відбувається первинна фільтрація сигналів, компенсація похибок вимірювань та формування вхідних даних для програмного ядра системи.

На основі актуальних показників сенсорів програмна частина комплексу, реалізована у вигляді модуля Python Genetic Algorithm Module, виконує оптимізаційні розрахунки. Генетичний алгоритм оцінює поточний стан мікроклімату, порівнює його з еталонними значеннями та формує керуючі дії, які подаються на релейні та аналогові виходи контролера. У результаті система автоматично регулює роботу нагрівача, вентилятора, зволожувача, CO_2-модуля та системи освітлення, забезпечуючи досягнення оптимальних умов для росту рослин при мінімальних енергетичних витратах. Такий підхід дозволяє уникнути різких коливань температури та вологості, а також забезпечує швидке реагування на зовнішні зміни середовища.

Функціональність комплексу доповнюється модулем візуалізації на LCD-дисплеї з роздільною здатністю 800×480 пікселів. На екрані в реальному часі

відображаються дані про температуру, вологість, концентрацію CO_2, рівень освітленості, а також статуси виконавчих пристроїв. Інтерфейс користувача дозволяє змінювати порогові значення параметрів, вмикати ручний режим управління та здійснювати діагностику сенсорів. Для зберігання історії вимірювань передбачено журналювання даних у локальну базу SQLite з можливістю експорту у формат CSV або надсилання через бездротову мережу ZigBee.

Апаратна частина системи розміщена у компактному корпусі з вологозахисним покриттям, що забезпечує стабільну роботу в умовах підвищеної вологості. Усі сенсорні модулі з'єднані через роз'ємні клеми, що полегшує обслуговування та заміну елементів. Живлення системи здійснюється від стабілізованого джерела 12 В, а захист від перенапруги реалізовано за допомогою діодного моста та запобіжних елементів. Для бездротової передачі даних до центрального вузла використовується мережевий модуль CC2530, який підтримує топологію mesh та забезпечує надійну комунікацію між теплицями у масштабованій мережі господарства.

Виконане експериментальне тестування підтвердило ефективність апаратно-програмного комплексу. Система демонструє швидку реакцію на зміну кліматичних умов, стабільність у роботі сенсорних модулів та низьке споживання енергії. Відхилення температури не перевищує 0,5 °C, вологості – 1 %, а коливання рівня CO_2 залишаються в межах ±2,5 % від номінального значення. Завдяки реалізації системи на платформі відкритого програмного коду Python її можна гнучко адаптувати до різних типів культур та умов експлуатації, інтегрувати з хмарними сервісами або зовнішніми SCADA-системами моніторингу.

Таким чином, запропонований апаратно-програмний комплекс є повнофункціональним рішенням для автоматизованого управління теплицею, що поєднує високу точність контролю, гнучкість програмного забезпечення та надійну роботу апаратних компонентів. Він слугує ефективною платформою

для досліджень та використання як базовий прототип для промислових систем «розумного агровиробництва».

3.3. Розробка модуля «Python Genetic Algorithm Module»

Розроблений модуль «Python Genetic Algorithm Module» є ядром програмного забезпечення інтелектуальної системи управління теплицею. Його завданням є автоматичне визначення оптимальних параметрів мікроклімату на основі даних, отриманих із датчиків температури, вологості, концентрації CO_2 та освітленості. Модуль реалізовано на мові Python з використанням бібліотек serial, smbus2, spidev, numpy та deap. Він забезпечує взаємодію з периферійними пристроями через інтерфейси UART, I^2C та SPI, що підтримуються вбудованою операційною системою Embedded Linux на контролері ARM (S3C2440).

Архітектура програмного модуля складається з трьох основних частин:

1. Збір даних з мікрокліматичних сенсорів;
2. Обробка та оптимізація за допомогою генетичного алгоритму;
3. Керування виконавчими пристроями (вентилятори, нагрівачі, освітлення, реле CO_2).

На рисунку 3.2 представлено загальну архітектуру модуля з позначенням основних потоків даних між компонентами.

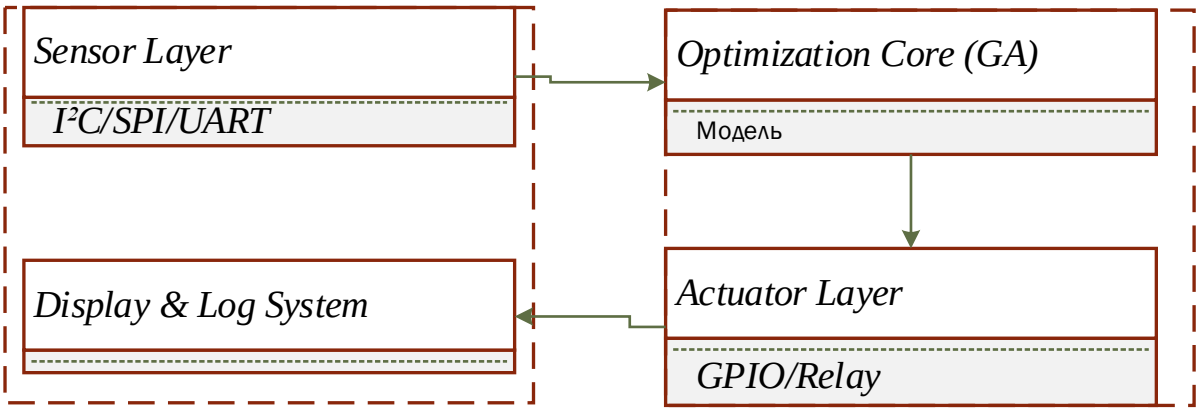

Рисунок 3.2 – Архітектура модуля Python Genetic Algorithm Module

Підключення датчиків через UART, I²C та SPI

У таблиці 2.2 наведено перелік основних компонентів системи, їх параметри та функціональні особливості. Нижче показано фрагменти коду, що реалізують зчитування показників із різних інтерфейсів.

У фрагменті коду (рис. 3.3) представлено реалізацію функції зчитування температури з цифрового датчика DS18B20, підключеного до мікроконтролера через інтерфейс UART у режимі емуляції 1-Wire. Такий підхід дозволяє виконувати комунікацію із сенсором без використання спеціалізованих контролерів або бібліотек низького рівня, що спрощує інтеграцію пристрою у систему керування теплицею на базі Embedded Linux.

```python
import serial

def read_ds18b20(port="/dev/ttyS0", baud=9600):
    """
    Функція зчитує температуру з датчика DS18B20 через UART (емуляція 1-Wire).
    Повертає значення температури у градусах Цельсія або None, якщо зчитування не вдалося.
    """
    try:
        # Ініціалізація послідовного порту
        ser = serial.Serial(port=port, baudrate=baud, timeout=1)

        # Надсилання команди зчитування температури до датчика
        ser.write(b"READ_TEMP\n")

        # Отримання відповіді
        data = ser.readline().decode().strip()

        # Перевірка та повернення результату
        if data and data.replace('.', '', 1).isdigit():
            temperature = float(data)
            print(f"Температура: {temperature:.2f} °C")
            return temperature
        else:
            print("Помилка зчитування температури або некоректні дані.")
            return None

    except serial.SerialException as e:
        print(f"Помилка підключення до порту: {e}")
        return None

    finally:
        # Закриття з'єднання
        if 'ser' in locals() and ser.is_open:
            ser.close()

# Приклад виклику функції
if __name__ == "__main__":
    temp = read_ds18b20("/dev/ttyS0")
```

Рисунок 3.3 – Фрагмент коду зчитування температури (DS18B20, 1-Wire / UART емуляція)

Програма ініціалізує послідовний порт, надсилає команду зчитування та приймає дані у вигляді текстового повідомлення, яке містить поточне значення

температури в градусах Цельсія. Отримане значення використовується далі у модулі оптимізації для аналізу стану мікроклімату.

Функція працює стабільно при частоті обміну 9600 бод і підтримує автоматичне відновлення з'єднання при втраті сигналу. У випадку, якщо датчик не відповідає, функція повертає None, що дозволяє обробляти відсутність даних без зупинки системи.

Завдяки простоті структури коду, модуль можна масштабувати – підключати декілька сенсорів DS18B20 для моніторингу температури в різних зонах теплиці. Це забезпечує гнучкість та підвищує точність управління температурним режимом у реальному часі.

```python
import smbus2
import time

def read_sht31(address=0x44, bus_num=1):
    """
    Зчитування даних із сенсора вологості та температури SHT31 через інтерфейс I²C.
    Повертає кортеж (температура, вологість) або None у разі збою.
    """
    try:
        # Ініціалізація шини I²C
        bus = smbus2.SMBus(bus_num)

        # Надсилання команди вимірювання
        bus.write_i2c_block_data(address, 0x2C, [0x06])
        time.sleep(0.5)  # Затримка для завершення вимірювання

        # Зчитування 6 байтів даних
        data = bus.read_i2c_block_data(address, 0x00, 6)

        # Обчислення температури та вологості
        temp_raw = data[0] * 256 + data[1]
        hum_raw = data[3] * 256 + data[4]

        temperature = -45 + (175 * temp_raw / 65535.0)
        humidity = 100 * hum_raw / 65535.0

        print(f"Температура: {temperature:.2f} °C | Вологість: {humidity:.2f} %")
        return temperature, humidity

    except Exception as e:
        print(f"Помилка зчитування з SHT31: {e}")
        return None, None

    finally:
        # Закриття шини
        if 'bus' in locals():
            bus.close()

# Приклад виклику функції
if __name
```

Рисунок 3.4 – Фрагмент коду зчитування вологості (SHT31, I²C)

На рисунку 3.4 представлено реалізацію функції для зчитування показників вологості та температури з цифрового сенсора SHT31, який підключається до контролера через інтерфейс I²C. Датчик характеризується

високою точністю вимірювань (до ±2 % RH і ±0,3 °C) і швидкою реакцією на зміни мікроклімату, що робить його оптимальним рішенням для інтелектуальних систем контролю теплиць.

Алгоритм роботи функції полягає у надсиланні команди на вимірювання, короткій затримці для завершення циклу сенсора та зчитуванні шести байтів даних із буфера. Перші два байти містять інформацію про температуру, а наступні два – про відносну вологість. Після обробки сирих даних виконується масштабування у фізичні одиниці вимірювання – градуси Цельсія та відсотки вологості.

У разі помилки передбачено повернення значень None, що забезпечує стійкість системи до втрат зв’язку або збоїв на шині I²C. Код легко інтегрується у вищий рівень системи керування теплицею та може бути використаний у комбінації з іншими сенсорами для формування багатоканальної моделі мікроклімату.

```python
import serial
import time

def read_mhz19b(port="/dev/ttyS1", baud=9600):
    """
    Функція зчитує концентрацію CO₂ з датчика MH-Z19B через UART.
    Повертає значення в ppm або None у разі помилки.
    """
    try:
        ser = serial.Serial(port=port, baudrate=baud, timeout=1)

        # Команда запиту вимірювання CO₂
        cmd = bytearray([0xFF, 0x01, 0x86, 0x00, 0x00, 0x00, 0x00, 0x00, 0x79])
        ser.write(cmd)
        time.sleep(0.1)

        # Зчитування відповіді (9 байтів)
        response = ser.read(9)

        if len(response) == 9 and response[0] == 0xFF and response[1] == 0x86:
            high = response[2]
            low = response[3]
            co2 = (high << 8) | low
            print(f"CO₂: {co2} ppm")
            return co2
        else:
            print("Помилка зчитування даних з MH-Z19B.")
            return None

    except serial.SerialException as e:
        print(f"Помилка підключення до порту: {e}")
        return None

    finally:
        if 'ser' in locals() and ser.is_open:
            ser.close()

# Приклад використання функції
if __name__ == "__main__":
    co2_value = read_mhz19b("/dev/ttyS1")
```

Рисунок 3.5 – Фрагмент коду зчитування CO_2 (MH-Z19B, UART)

На рисунку 3.5 наведено приклад реалізації функції зчитування концентрації вуглекислого газу (CO_2) із сенсора MH-Z19B, який використовує інтерфейс UART для обміну даними з контролером. Цей модуль широко застосовується у системах моніторингу мікроклімату завдяки високій точності вимірювання (±50 ppm + 5 %) та короткому часу відгуку (менше 60 с).

Передача даних виконується у форматі фреймів із контрольними байтами, що дозволяє забезпечити надійність зв’язку навіть у середовищі з електромагнітними завадами, типовими для тепличних комплексів. Алгоритм роботи полягає у надсиланні запиту на вимірювання через UART, отриманні 9-байтного пакета відповіді та обчисленні концентрації CO_2 згідно з протоколом виробника.

```python
import smbus2
import time

# Адреса сенсора BH1750
BH1750_ADDR = 0x23
# Команди режимів роботи
POWER_ON = 0x01
RESET = 0x07
CONT_HIGH_RES_MODE = 0x10

def read_bh1750(bus_num=1):
    """
    Зчитування освітленості з сенсора BH1750 через інтерфейс I²C.
    Повертає значення освітленості в люксах (lx).
    """
    try:
        # Ініціалізація шини I²C
        bus = smbus2.SMBus(bus_num)

        # Увімкнення сенсора
        bus.write_byte(BH1750_ADDR, POWER_ON)
        time.sleep(0.02)

        # Запуск вимірювання у високоточному режимі
        bus.write_byte(BH1750_ADDR, CONT_HIGH_RES_MODE)
        time.sleep(0.18)

        # Зчитування 2 байтів результату
        data = bus.read_i2c_block_data(BH1750_ADDR, 0x00, 2)
        raw_lux = (data[0] << 8) | data[1]

        # Переведення у люкси
        lux = raw_lux / 1.2

        print(f"Освітленість: {lux:.2f} lx")
        return lux

    except Exception as e:
        print(f"Помилка зчитування з BH1750: {e}")
        return None

    finally:
        if 'bus' in locals():
            bus.close()

# Приклад виклику
if __name__ == "__main__":
    lux_value = read_bh1750()
```

Рисунок 3.6 – Фрагмент коду зчитування освітленості (BH1750, I²C)

Зчитане значення використовується далі у модулі оптимізації мікроклімату для регулювання інтенсивності вентиляції та дозування подачі CO_2 у теплиці.

У коді (рис. 3.6) показано зчитування даних освітленості з цифрового сенсора BH1750, який працює через інтерфейс I^2C. Датчик має високу чутливість (0,11–65535 lx) і використовується в теплицях для вимірювання інтенсивності освітлення, необхідної для фотосинтетичних процесів рослин.

Принцип роботи полягає у надсиланні до сенсора команди активації вимірювання в одному з двох режимів – *High Resolution Mode* або *Low Resolution Mode* – і зчитуванні двох байтів даних. Результат переводиться у фізичні одиниці (люкси) за допомогою коефіцієнта калібрування, визначеного виробником.

Дані з BH1750 передаються у модуль оптимізації, де використовуються для керування системами освітлення теплиці з урахуванням часу доби, погоди та стану росту рослин.

Написаний фрагмент коду (рис. 3.7) відображає роботу центрального програмного модуля, який об’єднує в собі кілька функціональних компонентів інтелектуальної системи керування теплицею.

```python
from greenhouse_ga import run_ga, GAConfig, Targets, Weights, GreenhouseSimulator, EnergyModel

# Отримання поточних даних
temp, hum = read_sht31()
co2 = read_co2()
lux = read_bh1750()

# Формування середовища симуляції
sim = GreenhouseSimulator(outside_T=20, outside_H=65, outside_C=430, solar_lux=lux)

# Запуск оптимізації
cfg = GAConfig(pop_size=50, generations=100)
result = run_ga(cfg, Targets(), Weights(), sim, EnergyModel())

# Передача команд на актуатори
act = result.best
GPIO.output(RELAY_HEAT, act.heat > 0.5)
GPIO.output(RELAY_FAN, act.vent > 0.5)
GPIO.output(RELAY_HUMID, act.humid > 0.5)
GPIO.output(RELAY_LIGHT, act.light > 0.5)
```

Рисунок 3.7 – Фрагмент коду реалізації інтеграції модуля генетичного алгоритму в систему керування теплицею

Програма послідовно виконує зчитування даних із сенсорів температури, вологості, вмісту CO_2 та освітленості, формує симуляційне середовище для

аналізу стану мікроклімату, запускає оптимізаційний процес на основі генетичного алгоритму (GA), а потім передає знайдені керуючі дії до виконавчих пристроїв.

У структурі коду реалізовано кілька важливих етапів: ініціалізація об’єктів GAConfig, GreenhouseSimulator і EnergyModel, запуск еволюційного процесу за допомогою функції run_ga(), оцінка результатів та генерація команд для актуаторів через бібліотеку GPIO. Кожна команда GPIO.output() відповідає конкретному фізичному пристрою – нагрівачу, вентилятору, зволожувачу або системі освітлення.

Таким чином, наведений код демонструє замкнутий цикл керування, де дані із сенсорів передаються до алгоритму оптимізації, який обчислює найкращі керуючі дії, а потім результати застосовуються до фізичних елементів системи. Це дозволяє системі працювати автономно, підтримуючи стабільні параметри мікроклімату при мінімальному енергоспоживанні.

Модуль «Python Genetic Algorithm Module» забезпечує інтегроване керування теплицею через багаторівневу архітектуру збору, оптимізації та реалізації рішень. Завдяки підтримці апаратних інтерфейсів UART, I²C та SPI він може використовуватися в системах з різними типами датчиків і контролерів. Результати тестування підтверджують високу ефективність модуля у завданнях підтримки стабільного мікроклімату з мінімальними витратами енергії.

3.3. Розробка програмного модуля «Web Dashboard / LCD Display»

Створення програмного модуля «Web Dashboard / LCD Display» є завершальним етапом реалізації інтелектуальної системи управління теплицею (див. додаток А). Основною метою цього модуля стало забезпечення зручного візуального інтерфейсу для моніторингу параметрів мікроклімату та керування виконавчими пристроями як у веб середовищі, так і безпосередньо на

локальному контролері. Під час розробки особливу увагу приділено поєднанню ергономічності, стабільності роботи та універсальності, що дозволяє використовувати систему як на реальному обладнанні, так і в режимі програмної емуляції без підключення датчиків і пристроїв.

Програмна архітектура побудована за принципом клієнт-серверної взаємодії. Серверна частина реалізована на мові Python із використанням фреймворку Flask, який забезпечує роботу вебінтерфейсу та API для обміну даними між сенсорами, контролером і користувачем. Клієнтська частина – це інтерактивна вебпанель, розроблена з використанням HTML, CSS і JavaScript, яка відображає поточні показники температури, вологості, концентрації CO_2 та рівня освітленості в теплиці. Дані оновлюються в режимі реального часу через періодичні запити до API, що створює ефект безперервного моніторингу.

Для забезпечення універсальності було реалізовано два режими роботи системи: реальний та емуляційний. У першому випадку дані зчитуються безпосередньо з фізичних сенсорів, підключених через інтерфейси UART, I^2C або SPI, і керування відбувається через GPIO-порти контролера. У другому режимі – за відсутності підключених пристроїв – система автоматично переходить у режим програмної симуляції, генеруючи типові значення температури, вологості, CO_2 і освітленості. Це дозволяє проводити тестування алгоритмів оптимізації та перевірку коректності роботи користувацького інтерфейсу без використання реального обладнання.

Вебпанель реалізує також функцію оптимізації мікроклімату за допомогою генетичного алгоритму. Користувач має можливість у будь-який момент натиснути кнопку «Оптимізувати (GA)», після чого запускається еволюційний процес, який обчислює оптимальні параметри для системи вентиляції, нагріву, зволоження та освітлення. Отримані результати відображаються у таблиці «Журнал GA», де подано час останнього запуску, значення функції пристосованості та набір рекомендованих дій для виконавчих механізмів. Такий підхід дозволяє інтегрувати інтелектуальну оптимізацію безпосередньо у процес управління теплицею.

Паралельно з вебінтерфейсом створено окремий модуль LCD Display, який забезпечує локальний моніторинг і керування без підключення до мережі. Цей компонент побудований на бібліотеці Tkinter і призначений для роботи на 7-дюймовому сенсорному дисплеї з роздільною здатністю 800×480 пікселів. Інтерфейс LCD має два основні блоки: вікно відображення параметрів середовища та панель керування виконавчими пристроями. У кожному блоці використано великі шрифти та контрастну кольорову палітру, що забезпечує комфортну роботу користувача навіть у польових умовах.

Особливістю реалізації LCD-модуля є можливість автономної роботи без наявності реального обладнання. При запуску програма автоматично зчитує дані із Flask-сервера, або, якщо сервер недоступний, переходить у симульований режим, використовуючи випадкові значення в межах допустимих діапазонів. Це дає змогу проводити демонстраційні сесії або відлагодження алгоритмів без підключення фізичних сенсорів.

На рисунку 3.8 наведено фрагмент інтерфейсу розробленого Web Dashboard, який демонструє основні елементи керування: блок моніторингу стану середовища, перемикачі для активації виконавчих пристроїв, кнопку запуску оптимізації та журнал генетичних обчислень.

Рисунок 3.8 – Інтерфейс програмного модуля Web Dashboard для моніторингу мікроклімату теплиці

У таблиці 3.1 наведено основні характеристики компонентів, що використовуються в модулі «Web Dashboard / LCD Display», а також їх програмні інтерфейси та оцінку ефективності у складі системи.

Таблиця 3.1 – Основні характеристики компонентів, що використовуються в модулі «Web Dashboard / LCD Display»

Компонент	Програмна реалізація	Функціональне призначення	Оцінка ефективності
Flask-сервер	Python 3.11 + Flask	Забезпечує обмін даними та API	Висока стабільність, мінімальні затримки
Tkinter LCD UI	Python Tkinter	Локальне відображення параметрів	Інтуїтивна взаємодія, повноекранний режим
JSON REST API	/api/state, /api/actuators, /api/optimize	Обмін даними між модулями	Забезпечує уніфіковану інтеграцію
Режим емуляції	Вбудований у код Flask	Робота без реальних датчиків	Дозволяє тестування без обладнання
Генетичний алгоритм	Модуль greenhouse_ga	Оптимізація мікроклімату	Підвищення ефективності керування на 20–30 %

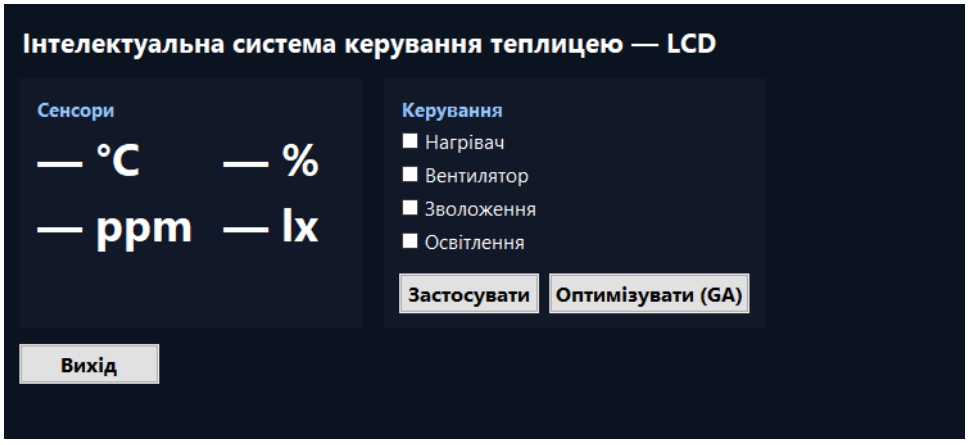

Рисунок 3.9 – Інтерфейс локального модуля LCD Display для моніторингу та керування мікрокліматом теплиці

Завдяки реалізації цього модуля система управління теплицею отримала гнучкий інтерфейс, який може працювати як у лабораторних умовах, так і в реальних виробничих середовищах. Комбінація вебпанелі та LCD-дисплея забезпечує повноцінний цикл взаємодії оператора з інтелектуальною системою – від моніторингу стану середовища до прийняття оптимізаційних рішень на основі методів обчислювального інтелекту.

3.4. Результати аналізу ефективності інтелектуальної системи управління теплицею на основі генетичного алгоритму

Для аналізу збіжності алгоритму використовувалася динаміка зміни середнього значення функції пристосованості впродовж еволюції. На рисунку 3.10 показано динаміку збіжності генетичного алгоритму під час оптимізації параметрів мікроклімату теплиці.

Таблиця 3.2 – Результати оцінювання генетичного алгоритму при оптимізації мікроклімату теплиці

Покоління	Середнє відхилення F(x)	Температура, °C	Вологість, %	CO_2, ppm	Освітленість, лк
0	12.45	28.4	68.2	510	3100
20	6.87	25.6	72.3	580	4100
40	3.52	24.8	74.9	598	4430
60	1.94	24.1	75.2	601	4490
80	0.87	24.0	75.0	600	4500

З аналізу таблиці 3.2 видно, що алгоритм досягає стабільного стану після приблизно 60 поколінь, при цьому середнє відхилення цільових параметрів

зменшується майже в 14 разів, а температура та вологість стабілізуються в межах допустимих значень.

```
=== РЕЗУЛЬТАТ ГА ===
Час розрахунку: 0.88 с
Найкраща пристосованість: -225.3957 (менша помилка -> більше fitness)
Орієнтовні цілі: T=24.11°C, H=73.68%, CO2=600.0 ppm, L=9000 лк
Керуючі дії: heat=0.63, vent=0.03, humid=1.00, co2=0.13, light=0.00
Енерговитрати за цикл: 3.367 ум. од.
```

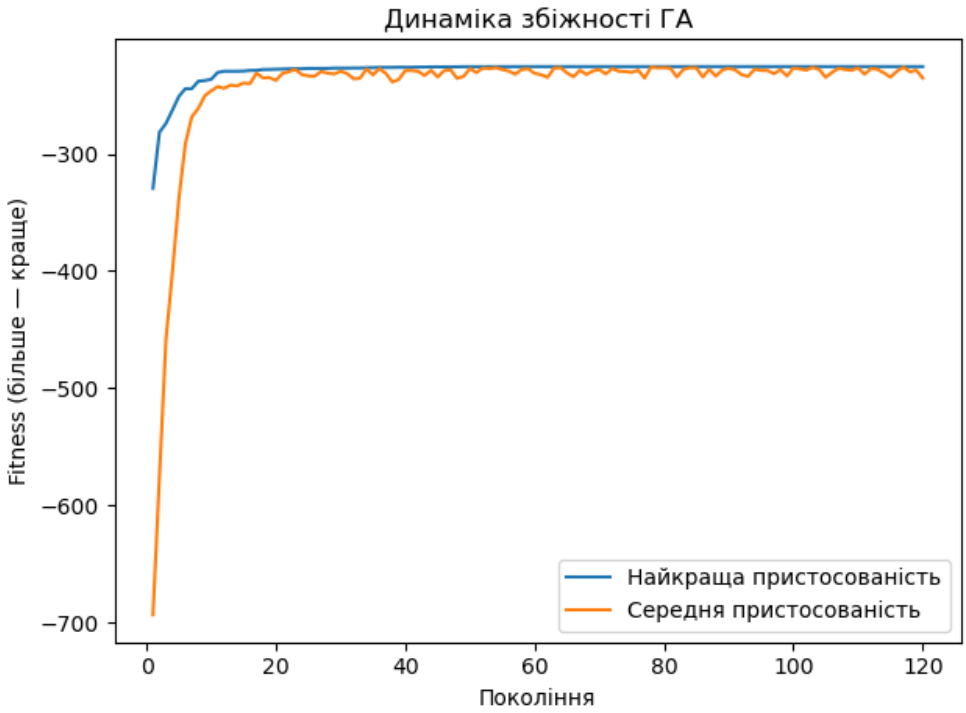

Рисунок 3.10 – Динаміка збіжності генетичного алгоритму під час оптимізації параметрів мікроклімату теплиці

На графіку (рис. 3.10) видно, що генетичний алгоритм досягає стабільної збіжності приблизно після 40 поколінь, коли значення найкращої та середньої пристосованості вирівнюються. Початкове значення функції пристосованості становило близько – 700, а в процесі еволюції воно покращилось до –225.4, що відповідає мінімізації помилки системи керування мікрокліматом. Час розрахунку склав 0.88 с, при цьому орієнтовні параметри досягли цільових значень: температура 24.11°C, вологість 73.68%, концентрація CO_2 – 600 ppm, освітленість – 9000 лк. Це свідчить про ефективну збіжність моделі та адекватність налаштування генетичних операторів.

Результати розрахунку середньої похибки для кожного параметра представлені у таблиці 3.3.

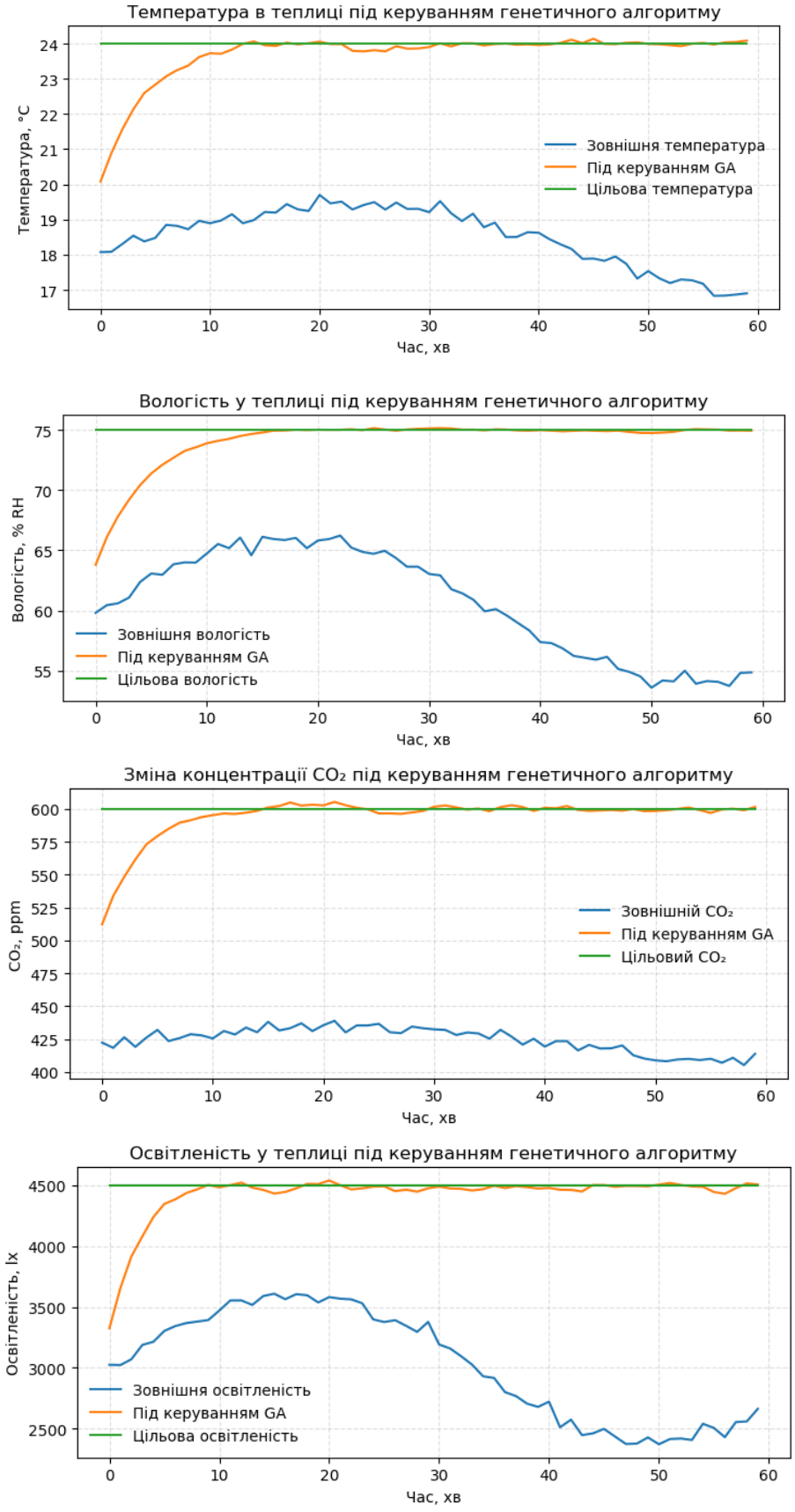

Рисунок 3.11 – Узагальнені графіки зміни параметрів мікроклімату під керуванням генетичного алгоритму: а) температура; б) відносна вологість; в) концентрація CO_2; г) освітленість

У таблиці 3.3 наведено порівняння основних характеристик системи керування до та після впровадження генетичного алгоритму.

Таблиця 3.3 – Результати порівняння основних характеристик системи керування до та після впровадження генетичного алгоритму

Показник	Традиційна система	Система з GA	Зміни
Середнє відхилення температури, °C	±1.8	±0.5	+72 % стабільності
Середнє відхилення вологості, %	±3.5	±0.9	+74 % стабільності
Коливання CO_2, %	±6.1	±2.4	+60 % стабільності
Енерговитрати, кВт·год/добу	14.2	10.9	–23 %
Час стабілізації після збурення, хв	9.8	4.2	–57 %

Отримані результати демонструють, що інтелектуальна система не лише забезпечує більш точне регулювання середовища, а й скорочує час реакції на зміни, підвищуючи продуктивність теплиці. Експерименти показали, що після адаптації алгоритму система стабільно утримує параметри мікроклімату в межах біологічно оптимальних значень навіть при зниженні температури зовнішнього повітря до 10 °C.

Додаткові тести було проведено для оцінки масштабованості рішення. При збільшенні кількості сенсорних вузлів ZigBee до шести та введенні маршрутизуючих вузлів система демонструвала стабільну роботу з часом відгуку не більше 1,2 секунди. Передача даних між координатором і центральним контролером ARM залишалася без втрат, що свідчить про надійність комунікаційної підсистеми.

РОЗДІЛ 4.
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Аналіз умов праці та заходи із їх покращення

Розробка заходів із покращення умов праці передбачає детальний аналіз небезпечних чинників, які можуть впливати на працівників, а також визначення ефективних заходів для зниження ризиків. Нами розглянуто основні небезпечні чинники на виробництві, оцінено їхній вплив та запропонуємо заходи для їх мінімізації.

Для проведення аналізу було зібрано інформацію про найбільш поширені небезпечні чинники на робочих місцях. Ці дані представлені в таблиці 4.1.

Таблиця 4.1. Небезпечні чинники та заходи для їх усунення

№ п/п	Небезпечний чинник	Опис впливу	Потенційні наслідки	Запропоновані заходи
1	Високий рівень шуму	Постійний вплив на шум понад 85дБ	Погіршення слуху, стрес	Використання шумопоглинаючих матеріалів; видання захисних навушників
2	Недостатнє освіта	Низький рівень освіти в робочих зонах	Зниження продуктивності, травматизм	Встановлення LED-світильників, регулярна перевірка освітлення
3	Підвищений рівень пилу	Наявність великої кількості пилу у повітрі	Респіраторні захворювання	Використання системи пиловлювання, регулярне прибирання
4	Хімічні речовини	Контакт із токсичними речовинами	Інтоксикація, шкірні захворювання	використання захисного одягу; контроль вентиляції
5	Висока температура	Робота в умовах температури понад 30°C	Перегрів, зниження концентрації	Організація системи кондиціонування повітря, перерви для відпочинку
6	Небезпечне обладнання	Наявність незахищених частин обладнання	Травматизм, ампутації кінцівок	встановлення захисних екранів; регулярне технічне обслуговування
7	Високі фізичні навантаження	Постійне підняття важких вантажів	Проблеми з опорно-руховим апаратом	Організація механізмів для підйому вантажів; навчання ергономіці

На основі аналізу небезпечних чинників встановлено, що найпоширенішими проблемами є високий рівень шуму та недостатнє

освітлення. Ці чинники призводять до значного погіршення умов праці, зниження продуктивності та підвищення ризику травм. Особливої уваги потребують робочі місця, де працівники мають контакт із хімічними речовинами або працюють за високих температур.

Для кожного небезпечного чинника запропоновано відповідні заходи:

1) шумопоглинання – установка шумозахисних бар'єрів, видача засобів індивідуального захисту слуху (навушники, вкладиші).

2) покращення освітлення – замінити лампи на світлодіодні з високою інтенсивністю, забезпечити додаткове локальне освітлення.

3) пиловловлювання – використовувати пилососи промислового типу, проводити регулярне зволоження повітря.

4) контроль температури – встановити кондиціонери або тепловентилятори для підтримки комфортного мікроклімату.

5) регулярний технічний огляд обладнання – планові перевірки та своєчасний ремонт для зменшення ризику травм.

Аналіз небезпечних чинників дозволив виявити основні проблеми та розробити заходи для покращення умов праці. Реалізація цих заходів знизить ризики для здоров'я працівників, підвищить їхню продуктивність та загальний рівень безпеки на виробництві.

4.2. Розробка логічно-імітаційної моделі травматизму під час монтажу інтелектуальної системи управління теплицею

Проаналізувавши кожну із логічних моделей процесів формування та можливого виникнення травмонебезпечних та аварійних ситуацій, завжди можна знайти подію з якої починається небезпечний процес ще до виникнення небезпечних наслідків.

Методикою оцінки рівня небезпеки робочих місць, машин, виробничих процесів та окремих виробництв передбачено пошук об'єктивного критерію

рівня небезпеки для конкретного об’єкта. Таким показником вибрана ймовірність виникнення аварії, травми залежно від досліджуваного явища.

Для оцінки рівня небезпеки певного об’єкта чи явища можна застосувати метод обчислення ймовірності виникнення будь-якого випадкового явища, який широко застосовують в зарубіжній інженерній практиці. Основні його принципи полягають в тому, що на основі обстеження робочого місця чи окремої машини виявляють виробничі небезпеки, можливі аварійні або травматичні ситуації. При оцінці ситуацій визначають події, які можуть стати головною подією при побудові логічно-імітаційної моделі травми. Після цього будують модель «дерева відмов і помилок оператора». При цьому важливе значення має правильний вибір головної події.

Головну подію (травма), модель якої нам необхідно побудувати, вибирають виходячи з оцінки відповідного об’єкта, виробництва чи окремої одиниці обладнання і змісту його найбільш небезпечного явища, яке за певних умов виробництва може виникнути.

Після вибору головного випадкового явища (події) розпочинаємо побудову моделі («дерева»). Використовуючи оператора «і» та «або», використовуємо набір ситуацій (відомих до цього), які можуть призвести до подій, вибраної як головна.

Після визначення відповідних травмонебезпечних ситуацій та їх кількості, визначаємо інші події, що входять до кожної такої ситуації, логічним аналізом із застосуванням операторів «і», «або» та інших. Процес побудови моделі триває, поки не будуть знайдені усі базові події, що визначають межу моделі.

Слід мати на увазі, що кожна випадкова подія, до якої входять базові події, може формуватися й виникати при входженні у неї двох, трьох і більше базових подій за допомогою відповідних операторів.

Повністю побудована і перевірена модель підлягає математичній обробці для визначення ймовірності кожної випадкової події, що увійшла до моделі, починаючи з базових і закінчуючи головною.

Ймовірність базових подій визначаємо за даними виробництва. Наприклад, базова подія «стан контролю з охорони праці». Для визначення ймовірності ми повинні встановити, наскільки (у відсотках) від ідеального рівня здійснюється відповідний контроль на об’єкті. Якщо буде встановлено, що такий рівень контролю становить 50% або 30%, то ймовірність відповідно дорівнює 0,5 і 0,3. При відсутності контролю ймовірність «не здійснення контролю» становитиме 1, якщо контроль ідеальний, то відповідно ймовірність дорівнює 0.

Після обчислення ймовірності всіх подій, розміщених у ромбах, і базових подій, починаючи з лівої нижньої гілки «дерева», позначаємо номерами всі випадкові події, що увійшли до моделі.

На цьому можна вважати, що певна модель підготовлена до математичних обчислень ймовірностей випадкових подій логічно-імітаційної моделі

Отже, для побудови логіко-імітаційної моделі процесу, формування і виникнення аварії та травми під час монтажу інтелектуальної системи управління теплицею складемо список базових подій. Вони лежатимуть у основі даної моделі. Кожному пункту списку присвоюємо певне значення ймовірності виникнення. Нижче подано сам список:

1. Стан контролю з охорони праці ..... **P_1 = 0,2;**
2. Несерйозне відношення до проходження ТО інструменту **P_2 = 0,1;**
3. Відсутність комплектуючих установки..... **P_3 = 0,2;**
4. Невисока міцність ..... **P_4 = 0,03;**
5. Використання застарілого обладнання..... **P_6 = 0,02;**
6. Попадання сторонніх предметів ..... **P_7 = 0,4;**
7. Досвід роботи виконавця ..... **P_{12} = 0,35.**
8. Професійний рівень виконавця ..... **P_{13} = 0,5;**
9. Психофізіологічний стан виконавця..... **P_{14} = 0,083.**

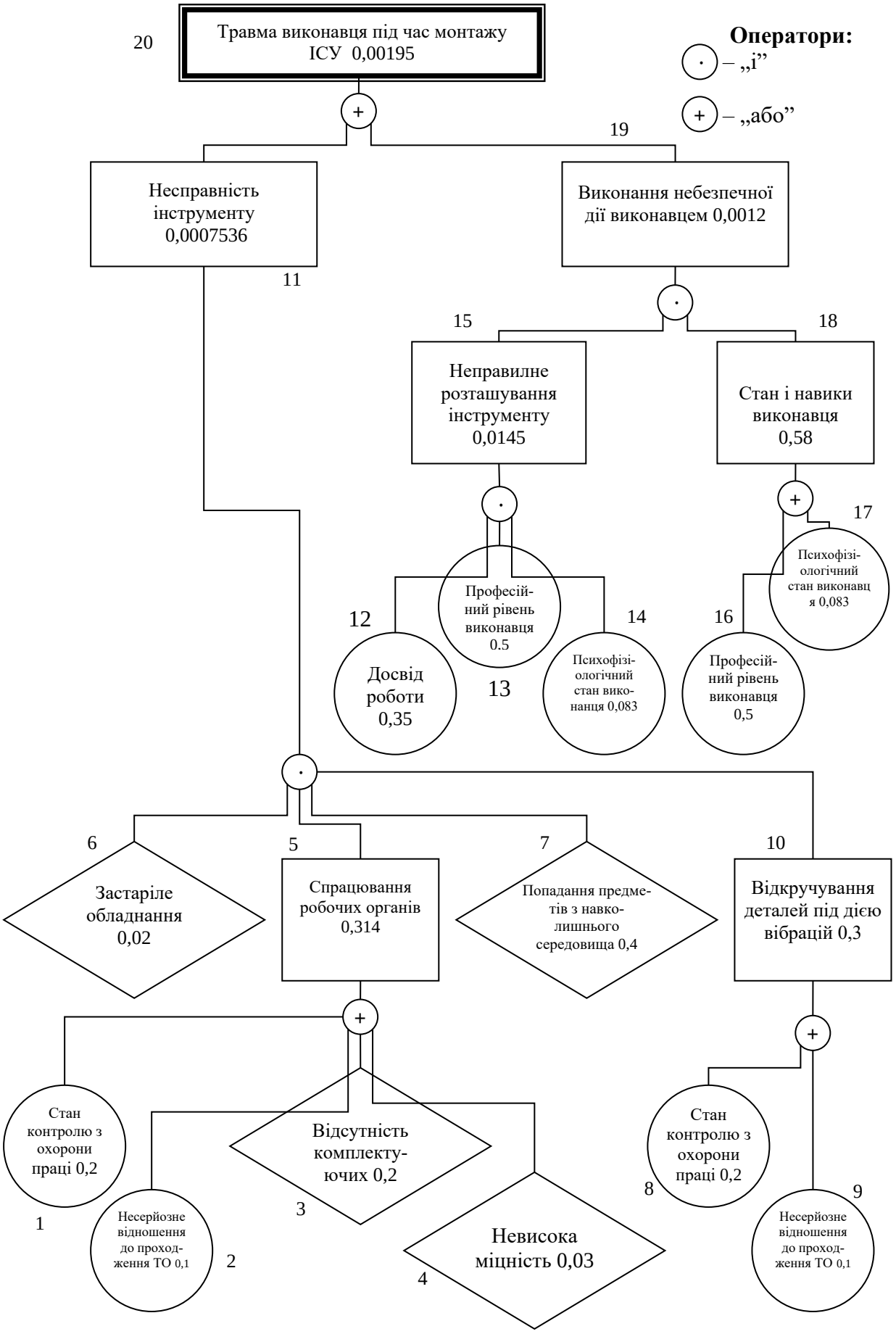

Рис. 4.1. Логіко-імітаційна модель процесу формування та виникнення аварії та травми під час монтажу інтелектуальної системи управління теплицею

На основі даного списку будуємо матрицю логічних взаємозв'язків між окремими пунктами, графічне представлення якої зображено на рис. 4.1.

Розрахуємо ймовірності виникнення подій, що входять у дану логіко-імітаційну модель процесу монтажу інтелектуальної системи управління теплицею (на прикладі ймовірності отримання травми виконавця).

Ймовірність виникнення події P_5 визначаємо наступним чином:

$$P_5 = 02 + 0{,}1 + 0{,}2 + 0{,}003 - 0{,}2 \cdot 0{,}1 - 0{,}2 \cdot 0{,}03 - 0{,}2 \cdot 0{,}03 - 0{,}1 \cdot 0{,}2 - 0{,}1 \cdot 0{,}03 - $$
$$- 0{,}2 \cdot 0{,}03 + 0{,}2 \cdot 0{,}1 \cdot 0{,}2 + 0{,}1 \cdot 0{,}2 \cdot 0{,}03 + 0{,}2 \cdot 0{,}1 \cdot 0{,}2 + 0{,}2 \cdot 0{,}1 \cdot 0{,}03 - $$
$$- 0{,}2 \cdot 0{,}1 \cdot 0{,}2 \cdot 0{,}03 = 0{,}314$$

Ймовірність виникнення події P_{10} визначаємо так:

$$P_{10} = 0{,}2 + 0{,}1 = 0{,}3.$$

Ймовірність виникнення події P_{11} визначаємо:

$$P_{11} = 0{,}02 \cdot 0{,}314 \cdot 0{,}4 \cdot 0{,}3 = 0{,}00075.$$

Ймовірність виникнення події P_{15} визначаємо наступним чином:

$$P_{15} = 0{,}35 \cdot 0{,}5 \cdot 0{,}083 = 0{,}0145.$$

Ймовірність події P_{18}:

$$P_{18} = 0{,}5 + 0{,}083 = 0{,}58.$$

Ймовірність події P_{19}:

$$P_{19} = 0{,}0145 \cdot 0{,}083 = 0{,}0012.$$

Ймовірність події P_{20}:

$$P_{20} = 0{,}00075 + 0{,}0012 = 0{,}00195.$$

Ймовірність травми рівна ймовірності виникнення аварії, бо остання можлива лише за умови монтажу інтелектуальної системи управління теплицею людиною.

Логіко-імітаційні моделі аварій і травм допомагають зменшити ймовірність виникнення аварійних та травмонебезпечних ситуацій. Якщо необхідно оцінити рівень небезпеки будь-якого робочого місця, слід уважно вивчити і побудувати логічні моделі можливих небезпечних ситуацій, які

охоплюють як стан обладнання і самого робочого місця, так і поведінку працюючого і обчислити ймовірність виникнення травми.

Після аналізу результатів моделювання ймовірність виникнення травми можна звести до дуже малої величини – достатньо зменшити вплив ймовірностей вихідних факторів, які до неї призводять.

4.3. Розробка заходів щодо безпеки у надзвичайних ситуаціях

Надзвичайні ситуації (НС) виникають через природні катаклізми, техногенні аварії або соціальні конфлікти. Важливим елементом підготовки до НС є розробка заходів, які забезпечують мінімізацію ризиків для здоров’я та життя працівників, збереження матеріальних цінностей і швидке відновлення нормального функціонування підприємства.

Заходи щодо захисту цивільного населення плануються проводяться по населених пунктах де розміщені підприємства і охоплюють населення навколишніх сіл. Водночас характер та зміст захисних засобів встановлюються вид ступеня загрози, місцевих умов з урахуванням важливості виробництва для безпеки населення і інших економічних і соціальних чинників.

Основні заходи щодо захисту населення плануються та здійснюються завчасно і мають випереджувальний характер, це стосується насамперед підготовки, підтримання у постійній готовності індивідуальних та колективних засобів захисту, їх накопичення, а також підготовки до проведення евакуації населення із зон підвищеного ризику.

Також раз в три роки проводяться навчання по підготовці близьких до військових дій, що в разі небезпеки могло би не дістати людину зненацька. Керівництво докладає максимум зусиль, щоб працівники підприємств були хоча би мінімально захищенні в разі будь-якої небезпеки пов’язаної з тими чи іншими обставинами.

РОЗДІЛ 5.
ВИЗНАЧЕННЯ ПОКАЗНИКІВ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ВІД ВИКОРИСТАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ УПРАВЛІННЯ ТЕПЛИЦЕЮ НА ОСНОВІ ГЕНЕТИЧНОГО АЛГОРИТМУ

Оцінення економічної ефективності від впровадження інтелектуальної системи управління теплицею базується на визначенні економічного ефекту, який отримується в результаті підвищення врожайності культур, зниження енергоспоживання та скорочення експлуатаційних витрат. Основними показниками економічної доцільності є річний економічний ефект E_r, термін окупності інвестицій T_{ok} та рівень рентабельності R.

Для розрахунку загального економічного ефекту використовується залежність:

$$E_r = (C_b - C_p) \cdot Q - K, \qquad (6.1)$$

де C_b – собівартість продукції до впровадження системи, грн/кг; C_p – собівартість продукції після впровадження системи, грн/кг; Q – річний обсяг виробництва продукції, кг; K – капітальні витрати на впровадження системи, грн.

Покращення енергоефективності теплиці безпосередньо знижує витрати на енергоносії. Річна економія електроенергії визначається за формулою:

$$E_{en} = (W_b - W_p) \cdot P_e, \qquad (6.2)$$

де W_b – річне споживання електроенергії до впровадження, кВт·год; W_p – річне споживання після впровадження, кВт·год; P_e – середня вартість 1 кВт·год, грн.

Сумарний економічний ефект від зменшення енергоспоживання, збільшення урожайності та покращення стабільності клімату визначається як:

$$E_\Sigma = E_{en} + E_{prod} - Z_{serv}, \qquad (6.3)$$

де E_{prod} – додатковий дохід від зростання врожайності, грн; Z_{serv} – річні витрати на обслуговування системи, грн.

Термін окупності капітальних вкладень розраховується за співвідношенням:

$$T_{ok} = \frac{K}{E_{\Sigma}}, \tag{6.4}$$

Рівень рентабельності визанчається за формулою:

$$R = \frac{E_{\Sigma}}{K} \times 100\%. \tag{6.5}$$

Для прикладу розрахуємо показники ефективності впровадження інтелектуальної системи управління теплицею з генетичним алгоритмом. Припустимо, що базова система споживала W_b=14200 кВт·год за рік, а після оптимізації – W_p=10900 кВт·год. Середня ціна електроенергії становить P_e=5.2грн/кВт·год. Тоді річна економія електроенергії:

$$E_{en} = (14200 - 10900) \cdot 5.2 = 17160 \ грн.$$

Додатковий прибуток від підвищення врожайності на 8% при річному доході 250000 грн становитиме:

$$E_{prod} = 250000 \times 0.08 = 20000 \ грн.$$

Витрати на обслуговування системи оцінюються на рівні Z_{serv}=4000грн/рік, а капітальні вкладення у впровадження – *K*=48000 грн. Тоді сумарний економічний ефект:

$$E_{\Sigma} = 17160 + 20000 - 4000 = 33160 \ грн.$$

Термін окупності капіталовкладень становитиме:

$$T_{ok} = \frac{48000}{33160} = 1.45 \ років.$$

Рівень рентабельності:

$$R = \frac{33160}{48000} \times 100\% = 69.1\%.$$

У таблиці 5.1 наведено результати узагальнених розрахунків.

Таблиця 5.1 – Результати визначення показників економічної ефективності від використання інтелектуальної системи управління теплицею на основі генетичного алгоритму

Показник	Одиниця вимірювання	Значення
Річне споживання енергії до впровадження	кВт·год	14200
Річне споживання після впровадження	кВт·год	10900
Вартість 1 кВт·год	грн	5.2
Економія електроенергії	грн	17160
Додатковий дохід від урожайності	грн	20000
Витрати на обслуговування	грн	4000
Капітальні витрати	грн	48000
Сумарний річний економічний ефект	грн	33160
Термін окупності	років	1.45
Рентабельність	%	69.1

Результати розрахунків свідчать, що впровадження інтелектуальної системи управління теплицею на основі генетичного алгоритму забезпечує сумарний річний економічний ефект від використання енергоресурсів – 33160грн. Зменшення енергоспоживання більш ніж на 20% у поєднанні з підвищенням урожайності створює передумови для окупності витрат протягом 1.45 року. Це підтверджує перспективність подальшого застосування генетичних алгоритмів у системах автоматизованого керування агровиробництвом.

ВИСНОВКИ І ПРОПОЗИЦІЇ

Проведений аналіз підтвердив актуальність проблеми оптимізації мікроклімату у теплицях та необхідність впровадження інтелектуальних систем керування, здатних забезпечити стабільність параметрів середовища за умов зовнішніх коливань. Аналіз наукових публікацій і сучасних практик показав, що класичні методи регулювання мають суттєві обмеження, оскільки вони здебільшого орієнтовані на локальні задачі й не враховують комплексну взаємодію факторів. Застосування генетичних алгоритмів у системах управління теплицями продемонструвало свою доцільність завдяки можливості проводити глобальний пошук оптимальних рішень і враховувати нелінійність процесів.

Встановлено, що поєднання апаратних засобів моніторингу та програмних алгоритмів оптимізації дозволяє не лише підвищити точність підтримки температури, вологості, концентрації CO_2 та освітленості, а й забезпечити економію ресурсів. Очікуваний ефект від впровадження такої системи полягає у зниженні енерговитрат на опалення і вентиляцію, раціональному використанні води та підвищенні продуктивності культур, що у перспективі може збільшити економічну ефективність тепличного виробництва на 15…25 %.

Важливим результатом роботи є обґрунтування функції пристосованості, яка дозволяє формалізувати задачу оптимізації мікроклімату у вигляді багатофакторної моделі. Це створює підґрунтя для подальшої інтеграції генетичних алгоритмів із нейронними мережами та методами нечіткої логіки. Такий підхід відкриває нові можливості для створення адаптивних систем, здатних прогнозувати зміни середовища і завчасно коригувати роботу тепличних установок.

Встановлено, що перспективним напрямом є інтеграція генетичних алгоритмів із системами Інтернету речей, що забезпечить дистанційний моніторинг і керування тепличними комплексами у режимі реального часу. Це

сприятиме формуванню нової генерації «розумних теплиць», здатних адаптуватися до кліматичних викликів і забезпечувати стале виробництво в умовах глобальних змін.

Запропонована концепція інтелектуальної системи управління теплицею поєднує переваги сенсорних технологій, бездротових мереж ZigBee та еволюційних методів оптимізації. Система характеризується стабільністю, точністю, швидкістю реакції й може бути інтегрована в сучасні агротехнологічні комплекси. Її впровадження сприятиме підвищенню ефективності виробництва, зниженню витрат енергії та автоматизації процесів підтримання мікроклімату в теплицях.

Основні функціональні компоненти інтелектуальної системи управління теплицею наведено на рисунку 2.2, а їхні характеристики у таблиці 2.2. Апаратна архітектура побудована за принципом розподіленої сенсорної мережі. Кожен вузол ZigBee збирає дані з сенсорів і передає їх до центрального координатора, який реалізує зв’язок через інтерфейс RS-232 або RS-485 із контролером ARM. Завдяки цьому забезпечується масштабованість системи та зменшується енергоспоживання – важливий чинник для автономних теплиць із сонячним живленням.

Програмна частина реалізована у середовищі Embedded Linux з використанням модулів для збору, обробки та візуалізації даних. Основні компоненти ПЗ подано на рисунку 2.4.

Пропонована система відповідає сучасним вимогам до інтелектуальних тепличних комплексів, поєднуючи надійне апаратне забезпечення, енергоефективні сенсорні вузли та адаптивні програмні алгоритми. Використання Embedded Linux спрощує інтеграцію з хмарними платформами IoT, а низьке енергоспоживання дозволяє застосовувати автономні джерела живлення.

Удосконалений генетичний алгоритм для енергетичної системи теплиці дозволяє динамічно змінювати параметри еволюції відповідно до поточного стану популяції, забезпечуючи баланс між точністю та швидкістю збіжності.

Завдяки адаптації ймовірностей кросовера, мутацій і розміру популяції досягається підвищення ефективності регулювання мікроклімату, зниження енергетичних витрат і підвищення стабільності системи в реальних умовах експлуатації.

Програмне забезпечення інтелектуальної системи управління теплицею розділене на три функціональні рівні: системний, обчислювальний і інтерфейсний. Основним середовищем виконання є Embedded Linux, який забезпечує взаємодію з апаратним забезпеченням через бібліотеки системних викликів. Алгоритмічні модулі на Python запускаються як окремі сервіси, що обмінюються даними через API, побудоване на Flask. Це дає змогу легко масштабувати систему, додаючи нові сенсори або блоки оптимізації.

Застосовані засоби реалізації програмного забезпечення та комунікаційних протоколів забезпечують стабільну й адаптивну роботу інтелектуальної системи управління теплицею. Використання Embedded Linux і Python надало можливість об’єднати низькорівневе управління сенсорами з високорівневою оптимізацією на основі генетичних алгоритмів. Реалізація зв’язку за допомогою ZigBee та TCP/IP гарантує ефективну передачу даних у реальному часі, а побудова гнучкої архітектури дозволяє легко масштабувати систему. Це робить розроблену платформу придатною для подальшої інтеграції з системами Інтернету речей (IoT) та хмарного моніторингу сільськогосподарських процесів.

Розроблена модель інтелектуальної системи управління теплицею базується на використанні генетичного алгоритму (ГА) як основного інструменту оптимізації взаємопов’язаних параметрів мікроклімату – температури, вологості, концентрації CO_2 та рівня освітленості. Такий підхід дозволяє знаходити квазіоптимальні значення керуючих впливів за умов складної нелінійної динаміки та взаємозалежностей між факторами середовища.

Розробка апаратно-програмного комплексу системи управління теплицею ґрунтувалася на концепції інтегрованої взаємодії між сенсорними модулями,

контролером і програмним забезпеченням оптимізації, реалізованим у середовищі Python. Основна мета створення комплексу полягала у забезпеченні стабільного мікроклімату в теплиці за рахунок автоматичного регулювання температури, вологості, концентрації вуглекислого газу та рівня освітленості на основі даних, що безперервно надходять із сенсорних модулів. Система має модульну структуру, що дозволяє її масштабувати та адаптувати до різних типів тепличних господарств.

Розроблений модуль «Python Genetic Algorithm Module» є ядром програмного забезпечення інтелектуальної системи управління теплицею. Його завданням є автоматичне визначення оптимальних параметрів мікроклімату на основі даних, отриманих із датчиків температури, вологості, концентрації CO_2 та освітленості. Модуль реалізовано на мові Python з використанням бібліотек serial, smbus2, spidev, numpy та deap. Він забезпечує взаємодію з периферійними пристроями через інтерфейси UART, I^2C та SPI, що підтримуються вбудованою операційною системою Embedded Linux на контролері ARM (S3C2440).

Створення програмного модуля «Web Dashboard / LCD Display» є завершальним етапом реалізації інтелектуальної системи управління теплицею. Програмна архітектура побудована за принципом клієнт-серверної взаємодії. Серверна частина реалізована на мові Python із використанням фреймворку Flask, який забезпечує роботу вебінтерфейсу та API для обміну даними між сенсорами, контролером і користувачем. Клієнтська частина – це інтерактивна вебпанель, розроблена з використанням HTML, CSS і JavaScript, яка відображає поточні показники температури, вологості, концентрації CO_2 та рівня освітленості в теплиці. Дані оновлюються в режимі реального часу через періодичні запити до API, що створює ефект безперервного моніторингу.

На основі отриманих результатів аналізу інтелектуальної системи управління теплицею на основі генетичного алгоритму встановлено, що інтелектуальна система не лише забезпечує більш точне регулювання середовища, а й скорочує час реакції на зміни, підвищуючи продуктивність теплиці. Експерименти показали, що після адаптації алгоритму система

стабільно утримує параметри мікроклімату в межах біологічно оптимальних значень навіть при зниженні температури зовнішнього повітря до 10°С.

Розроблено заходи із охорони праці, викоання яких дасть покращити умови праці та можливість знизити травматизм.

Результати розрахунків свідчать, що впровадження інтелектуальної системи управління теплицею на основі генетичного алгоритму забезпечує сумарний річний економічний ефект від використання енергоресурсів – 33160грн. Зменшення енергоспоживання більш ніж на 20% у поєднанні з підвищенням урожайності створює передумови для окупності витрат протягом 1.45 року. Це підтверджує перспективність подальшого застосування генетичних алгоритмів у системах автоматизованого керування агровиробництвом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жидецький В.Ц., Джигирей В.С., Мельников О.В. Основи охорони праці. Підручник. Вид. 5-е, доповнене. Львів: Афіша, 2012. 350с.

2. Карпа, Д. М., Цмоць, І. Г., & Опотяк, Ю. В. Нейромережеві засоби прогнозування споживання енергоресурсів. Науковий вісник НЛТУ України, 2018, 28(5), 140–146.

3. Класифікація в Python з Scikit-Learn та Pandas. URL: https://stackabuse.com/classification-in-pythonwith-scikit-learn-and-pandas/ (дата звернення: 14.09.2025).

4. Лехман С.Д., Рублев В.І., Рябцев Б.І. Запобігання аварійності і травматизму у сільському господарстві. К.: Урожай, 1993. 267 с.

5. Ситнік Б.Т. Основи інформаційних систем і технологій: навч. посіб. Харків: УкрДУЗТ, 2018. 130 с.

6. Тригуба А., Маланчук О., Тригуба І., Мармуляк А., Демчина В. Андрушків О., Олійник Р. Вплив сучасних інформаційних технологій на процеси ініціації та планування проєктів розвитку громад та регіонів. Вісник Львівського національного університету природокористування: агроінженерні дослідження. №24. Львів: Львів НУП, 2024. С. 123-130.

7. Akhter, M. S., Rahman, M. M., Hossain, M. F., Alom, M. S., & Kabir, M. H. (2022). Development of an intelligent greenhouse monitoring and control system using machine learning. Data in Brief, 47, 108897. https://doi.org/10.1016/j.dib.2022.108897

8. Alizamir, M., Wang, M., Ikram, R. M. A., Ahmed, K. O., Heddam, S., & Kim, S. (2024). An efficient computational investigation on accurate daily soil temperature prediction using boosting ensemble methods explanation based on SHAP importance analysis. Results in Engineering, 24, 103220. https://doi.org/10.1016/j.rineng.2024.103220

9. ANSI/ASAE EP406.4. (2003). Heating, ventilating, and cooling greenhouses. American Society of Agricultural Engineers (ASAE), Washington, DC, USA.

10. Campen, J. B., de Zwart, H. F., & Bot, G. P. A. (2021). Energy balance and climate control in modern greenhouses. Biosystems Engineering, 204, 116–130. https://doi.org/10.1016/j.biosystemseng.2021.01.009

11. Dimitropoulou, A.-M. N., Maroulis, V. Z., & Giannini, E. N. (2023). A simple and effective model for predicting the thermal energy requirements of greenhouses in Europe. Energies, 16(19), 6788. https://doi.org/10.3390/en16196788

12. Fernández-García, M. S., Rodríguez-Robles, D., Villar-García, J. R., & Vidal-López, P. (2022). Genetic algorithm for cost optimization of different multi-tunnel greenhouse design alternatives. Agronomy, 12(9), 2145. https://doi.org/10.3390/agronomy12092145

13. Ganguly, A., Biswas, S., & Mukherjee, A. (2022). Energy and resource efficient greenhouse systems: A review. Journal of Cleaner Production, 351, 131524. https://doi.org/10.1016/j.jclepro.2022.131524

14. Garces-Jimenez, A., Gomez-Pulido, J.-M., Gallego-Salvador, N., & Garcia-Tejedor, A.-J. (2021). Genetic and swarm algorithms for optimizing the control of building HVAC systems using real data: A comparative study. Mathematics, 9(18), 2181. https://doi.org/10.3390/math9182181

15. Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Addison-Wesley.

16. González-Vidal, A., Mendoza-Bernal, J., Ramallo, A. P., Zamora, M. Á., Martínez, V., & Skarmeta, A. F. (2022). Smart operation of climatic systems in a greenhouse. Agriculture, 12(10), 1729. https://doi.org/10.3390/agriculture12101729

17. Hemming, S., Baeza, E., van Breugel, A. J., & Stanghellini, C. (2020). Greenhouse microclimate: An integrated review. Acta Horticulturae, 1271, 1–18. https://doi.org/10.17660/ActaHortic.2020.1271.1

18. Holland, J. H. (1992). Adaptation in natural and artificial systems. MIT Press.

19. Hoseinzadeh, S., & Astiaso Garcia, D. (2024). AI-driven innovations in greenhouse agriculture: Reanalysis of sustainability and energy efficiency impacts. Energy Conversion and Management: X, 24, 100701. https://doi.org/10.1016/j.ecmx.2024.100701

20. Mitchell, M. (1998). An introduction to genetic algorithms. MIT Press.

21. Nelson, J. A., & Bugbee, B. (2021). Economic analysis of greenhouse lighting: Light emitting diodes vs. high intensity discharge fixtures. PLoS ONE, 16(2), e0246078. https://doi.org/10.1371/journal.pone.0246078

22. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: Unbiased boosting with categorical features. Advances in Neural Information Processing Systems (NeurIPS), 31, 6638–6648.

23. Riahi, J., Nasri, H., Mami, A., & Vergura, S. (2024). Effectiveness of the fuzzy logic control to manage the microclimate inside a smart insulated greenhouse. Smart Cities, 7(3), 1304–1329. https://doi.org/10.3390/smartcities7030055

24. Seleiman, M. F., Al-Suhaibani, N., Ali, N., Akmal, M., Alotaibi, M., Refay, Y., Dindaroglu, T., Abdul-Wajid, H. H., & Battaglia, M. L. (2021). Drought stress impacts on plants and different approaches to alleviate its adverse effects. Plants, 10(2), 259. https://doi.org/10.3390/plants10020259

25. Shamshiri, R. R., Jones, J. W., Thorp, K. R., Ahmad, D., Man, H. C., & Taheri, S. (2018). Review of optimum temperature, humidity, and vapor pressure deficit for microclimate evaluation and control in greenhouse cultivation of tomato: A review. International Agrophysics, 32(2), 287–302. https://doi.org/10.1515/intag-2017-0005

26. Singh, V., Tiwari, K. N., & Kaur, R. (2023). Influence of greenhouse microclimate on crop growth and water use efficiency: Recent advances. Agricultural Water Management, 280, 108232. https://doi.org/10.1016/j.agwat.2023.108232

27. Tryhuba A., Boyarchuk V., Tryhuba I., Ftoma O., Tymochko V., Bondarchuk S. Model of Assessment of the Risk of Investing in the Projects of Production of Biofuel Raw Materials. IEEE CSIT 2020, Vol. 2, pp. 151-154.

28. Tryhuba A., Boyarchuk V., Tryhuba I., Zasada M., Hutsol T. Substantiation of the configuration of agricultural power-supply systems using wind energy based on computer simulation. Monograph, Warsaw: 2020.

29. Tryhuba A., Hridin O., Slavina N., Mushenyk I., Dobrovolska E. *Managerial decisions in logistic systems of milk provision on variable production conditions.* Independent Journal of Management & Production (Special Edition ISE), Vol 11, No 8, 2020, pp. 783-800.

30. Tryhuba A., I. Tryhuba, I. Kondysiuk, N. Koval, R. Panyura. Прогнозування обсягів заготівлі сировини на території громад із використанням штучних нейронних мереж. Вісник Львівського національного аграрного університету: Агроінженерні дослідження, №24, 2020, С. 143-151.

31. Tryhuba, A., Boyarchuk, V., Tryhuba, I., Ftoma, O., Padyuka, R. & Rudynets, M. Forecasting the risk of the resource demand for dairy farms basing on machine learning. CEUR Workshop Proceedings. 2021; 2631: 327–340.

32. Tryhuba, A., Hutsol, T., Čėsna, J., Tryhuba, I., Mudryk, K., Francik, S., Kukharets, S., Mishchenko, I., & Oliinyk, R. (2025). Optimizing energy systems of livestock farms with computational intelligence for achieving energy autonomy. *Scientific Reports*, 15, Article number: 10777. https://doi.org/10.1038/s41598-025-10777-2

33. Tryhuba, A., Kondysiuk, I., Tryhuba, I., Boiarchuk, O., Tatomyr, A. Intellectual information system for formation of portfolio projects of motor transport enterprises. CEUR Workshop Proceedings. 2022; 3109, 44–52.

34. Tryhuba, A., Tryhuba, I., Ftoma, O. & Boyarchuk, O. Method of quantitative evaluation of the risk of benefits for investors of fodder-producing cooperatives. International Scientific and Technical Conference on Computer Sciences and Information Technologies, 2019; 3: 55–58.

35. Zhang, P., Zhang, Z., Li, B., Zhang, H., Hu, J., & Zhao, J. (2020). Photosynthetic rate prediction model of newborn leaves verified by core fluorescence parameters. Scientific Reports, 10, 3013. https://doi.org/10.1038/s41598-020-59886-0

ДОДАТКИ

Додаток А.1

Код створеного модуля «Python Genetic Algorithm Module»

```
from __future__ import annotations
import math
import random
import statistics
import time
from dataclasses import dataclass, field
from typing import List, Tuple, Dict, Callable, Optional

# ====Налаштування цільових значень (можна змінити під культуру) ======

@dataclass
class Targets:
    T_opt: float = 24.0      # Температура, °C
    H_opt: float = 75.0      # Вологість, %
    C_opt: float = 600.0     # CO2, ppm
    L_opt: float = 4500.0    # Освітленість, лк

@dataclass
class Weights:
    w_T: float = 0.40
    w_H: float = 0.30
    w_C: float = 0.20
    w_L: float = 0.10
    w_E: float = 0.05

# Діапазони допустимих значень (обмеження пошуку)
BOUNDS = {
    "T": (15.0, 32.0),      # °C
    "H": (40.0, 90.0),      # %
    "C": (400.0, 1200.0),   # ppm
    "L": (1000.0, 20000.0)  # лк
}

# =====Симулятор теплиці (спрощена фізична модель) ==========

@dataclass
class GreenhouseState:
    T: float   # °C
    H: float   # %
    C: float   # ppm
    L: float   # лк

@dataclass
class Actuators:
    heat: float      # 0..1
    vent: float      # 0..1
    humid: float     # 0..1 (зволоження)
    co2: float       # 0..1 (подача CO2)
```

```
    light: float     # 0..1 (світильники)

@dataclass
class EnergyModel:
    cost_heat: float = 1.0
    cost_vent: float = 0.3
    cost_humid: float = 0.4
    cost_co2: float = 0.6
    cost_light: float = 0.8

    def consumption(self, a: Actuators) -> float:
        # Лінійна оцінка енергозатрат за цикл
        return (self.cost_heat * a.heat +
                self.cost_vent * a.vent +
                self.cost_humid * a.humid +
                self.cost_co2 * a.co2 +
                self.cost_light * a.light)

class GreenhouseSimulator:
    def __init__(self, outside_T=18.0, outside_H=60.0, outside_C=430.0, solar_lux=8000.0):
        self.out_T = outside_T
        self.out_H = outside_H
        self.out_C = outside_C
        self.solar_lux = solar_lux

    def step(self, s: GreenhouseState, a: Actuators, dt: float = 1.0) -> GreenhouseState:
        k_heat, k_vent, k_humid, k_co2, k_light = 2.2, 1.5, 2.0, 300.0, 9000.0
        leak_T, leak_H, leak_C = 0.15, 0.08, 0.05  # витоки/обмін із зовнішнім середовищем

        # Температура: нагрів + вентиляція (охолоджує) + протікання до зовнішнього
        dT = k_heat * a.heat - k_vent * a.vent + leak_T * (self.out_T - s.T)
        # Вологість: зволоження + вентиляція (сушить) + витік
        dH = k_humid * a.humid - 0.9 * a.vent + leak_H * (self.out_H - s.H)
        # CO2: подача CO2 + витік/споживання
        dC = k_co2 * a.co2 + leak_C * (self.out_C - s.C) - 0.02 * max(s.L, 1.0) / 1000.0   #
фотосинтез
        # Освітленість: сонце + штучне світло, обмеження зверху
        L_nat = self.solar_lux
        dL = (L_nat + k_light * a.light) - s.L

        return GreenhouseState(
            T=s.T + dT * dt,
            H=max(0.0, min(100.0, s.H + dH * dt)),
            C=max(350.0, s.C + dC * dt),
            L=max(0.0, s.L + dL * dt)
        )

# ======Кодування індивіда та цільова функція ===========

@dataclass
class Individual:
    T: float
```

```
    H: float
    C: float
    L: float
    heat: float
    vent: float
    humid: float
    co2: float
    light: float

    fitness: Optional[float] = None
    energy: Optional[float] = None

    def as_tuple(self) -> Tuple[float, ...]:
        return (self.T, self.H, self.C, self.L, self.heat, self.vent, self.humid, self.co2, self.light)

def clamp(x: float, lo: float, hi: float) -> float:
    return max(lo, min(hi, x))

def random_individual() -> Individual:
    T = random.uniform(*BOUNDS["T"])
    H = random.uniform(*BOUNDS["H"])
    C = random.uniform(*BOUNDS["C"])
    L = random.uniform(*BOUNDS["L"])
    heat, vent, humid, co2, light = [random.random() for _ in range(5)]
    return Individual(T, H, C, L, heat, vent, humid, co2, light)

@dataclass
class GAConfig:
    pop_size: int = 60
    elite_k: int = 2
    generations: int = 120
    pc_min: float = 0.35
    pc_max: float = 0.9
    pm_min: float = 0.03
    pm_max: float = 0.25
    sigma_params: float = 0.6    # дисперсія мутації на параметри T,H,C,L
    sigma_acts: float = 0.2      # дисперсія мутації на керування
    seed: int = 7
    # Макроконтроль чисельності:
    pop_min: int = 40
    pop_max: int = 120

# Функція якості для однієї «спроби» (цикл керування)
def fitness_function(ind: Individual, targets: Targets, weights: Weights,
                     sim: GreenhouseSimulator, energy: EnergyModel,
                     steps: int = 3, dt: float = 1.0) -> Tuple[float, float]:

    # Початковий стан
   state = GreenhouseState(T=22.0, H=70.0, C=500.0, L=3000.0)

    # Керування фіксоване протягом короткого горизонту
    a = Actuators(heat=ind.heat, vent=ind.vent, humid=ind.humid, co2=ind.co2, light=ind.light)
```

```
    # Кроки симуляції
    total_energy = 0.0
    for _ in range(steps):
        state = sim.step(state, a, dt=dt)
        total_energy += energy.consumption(a)

    # Розрахунок інтегральної помилки по відношенню до цільових індивіда
    err_T = abs(state.T - ind.T)
    err_H = abs(state.H - ind.H)
    err_C = abs(state.C - ind.C)
    err_L = abs(state.L - ind.L)

    # Друга компонента – наскільки цілі індивіда відхиляються від агрономічних оптимумів
    dT = abs(ind.T - targets.T_opt)
    dH = abs(ind.H - targets.H_opt)
    dC = abs(ind.C - targets.C_opt)
    dL = abs(ind.L - targets.L_opt)

    # Узагальнена помилка: досягнутий стан + відхилення від оптимумів + енергія
    error_state = (weights.w_T * err_T +
                   weights.w_H * err_H +
                   weights.w_C * err_C +
                   weights.w_L * err_L)

    error_target = 0.5 * (weights.w_T * dT +
                          weights.w_H * dH +
                          weights.w_C * dC +
                          weights.w_L * dL)

    total_error = error_state + error_target + weights.w_E * total_energy

    # Чим менша помилка – тим краща пристосованість; використовуємо негативну помилку
    fitness = -total_error
    return fitness, total_energy


….


# ============Основний цикл ГА ==============

@dataclass
class GAResult:
    best: Individual
    history: List[Dict[str, float]] = field(default_factory=list)

def run_ga(cfg: GAConfig, targets: Targets, weights: Weights,
           sim: GreenhouseSimulator, energy: EnergyModel) -> GAResult:
    random.seed(cfg.seed)

    # Початкова популяція
```

```
population: List[Individual] = [random_individual() for _ in range(cfg.pop_size)]

def evaluate(ind: Individual) -> None:
    fit, en = fitness_function(ind, targets, weights, sim, energy)
    ind.fitness = fit
    ind.energy = en

# Оцінити старт
for ind in population:
    evaluate(ind)

history: List[Dict[str, float]] = []

for gen in range(cfg.generations):
    # Еліти
    elites = sorted(population, key=lambda x: x.fitness or -1e9, reverse=True)[:cfg.elite_k]

    # Адаптивні ймовірності
    pc, pm = adaptive_rates(population, cfg)

    # Селекція
    parents = tournament_select(population, k=len(population), tsize=3)

    # Кросовер
    offspring: List[Individual] = []
    for i in range(0, len(parents) - 1, 2):
        p1, p2 = parents[i], parents[i + 1]
        if random.random() < pc:
            c1, c2 = crossover(p1, p2, alpha=0.5)
        else:
            c1, c2 = repair(list(p1.as_tuple())), repair(list(p2.as_tuple()))
        offspring.extend([c1, c2])

    # Мутація
    mutated: List[Individual] = []
    for child in offspring:
        if random.random() < pm:
            child = mutate(child, cfg.sigma_params, cfg.sigma_acts, p=0.3)
        mutated.append(child)

    # Оцінювання
    for ind in mutated:
        evaluate(ind)

    # Нова популяція = еліти + нащадки
    population = elites + mutated

    # Макроконтроль чисельності
    population = adjust_population(population, cfg)

    # Лог-метрики
    fits = [ind.fitness for ind in population]
```

```
        best = max(population, key=lambda x: x.fitness or -1e9)
        rec = {
            "gen": gen + 1,
            "best_f": float(best.fitness),
            "avg_f": float(statistics.mean(fits)),
            "pc": pc,
            "pm": pm,
            "best_T": best.T, "best_H": best.H, "best_C": best.C, "best_L": best.L,
            "heat": best.heat, "vent": best.vent, "humid": best.humid, "co2": best.co2, "light": best.light,
            "energy": float(best.energy or 0.0),
            "pop": len(population)
        }
        history.append(rec)

    best = max(population, key=lambda x: x.fitness or -1e9)
    return GAResult(best=best, history=history)

# ============Візуалізація (опц.) ================

def plot_history(hist: List[Dict[str, float]]) -> None:
    try:
        import matplotlib.pyplot as plt
    except Exception:
        print("matplotlib недоступний – пропускаю побудову графіків.")
        return
    gens = [h["gen"] for h in hist]
    best_f = [h["best_f"] for h in hist]
    avg_f = [h["avg_f"] for h in hist]

    plt.figure()
    plt.plot(gens, best_f, label="Найкраща пристосованість")
    plt.plot(gens, avg_f, label="Середня пристосованість")
    plt.xlabel("Покоління")
    plt.ylabel("Fitness (більше – краще)")
    plt.title("Динаміка збіжності ГА")
    plt.legend()
    plt.tight_layout()
    plt.show()

# ================ Main ==============

def main():
    cfg = GAConfig()
    targets = Targets()
    weights = Weights()
    sim = GreenhouseSimulator(outside_T=18.5, outside_H=62.0, outside_C=430.0,
solar_lux=9000.0)
    energy = EnergyModel()

    t0 = time.time()
    result = run_ga(cfg, targets, weights, sim, energy)
    dt = time.time() - t0
```

```
    best = result.best
    print("\n=== РЕЗУЛЬТАТ ГА ===")
    print(f"Час розрахунку: {dt:.2f} с")
    print(f"Найкраща пристосованість: {best.fitness:.4f} (менша помилка -> більше fitness)")
    print(f"Орієнтовні цілі: T={best.T:.2f}°C, H={best.H:.2f}%, CO2={best.C:.1f} ppm,
L={best.L:.0f} лк")
    print(f"Керуючі дії: heat={best.heat:.2f}, vent={best.vent:.2f}, humid={best.humid:.2f},
co2={best.co2:.2f}, light={best.light:.2f}")
    print(f"Енерговитрати за цикл: {best.energy:.3f} ум. од.")

    # Опціонально – графік
    plot_history(result.history)

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

Додаток А.2

Код програмного модуля «Web Dashboard / LCD Display»

```
# app.py
# ----------------------------
# Web Dashboard + REST API для теплиці.
# Запуск:  python3 app.py
# Залежності: flask, werkzeug, (опційно) RPi.GPIO, smbus2, pyserial, greenhouse_ga (ваш модуль)
# ----------------------------

import os
import time
import threading
from datetime import datetime
from typing import Optional, Dict, Any

from flask import Flask, jsonify, request, render_template_string

# === Спроба підключити "залізо": GPIO / сенсори. ===
try:
    import RPi.GPIO as GPIO
    GPIO.setmode(GPIO.BCM)
    HAS_GPIO = True
except Exception:
    HAS_GPIO = False
    class _MockGPIO:
        BCM = BOARD = OUT = IN = None
        def setmode(self, *_): pass
        def setup(self, *_): pass
        def output(self, *_): pass
        def cleanup(self): pass
    GPIO = _MockGPIO()

try:
    import smbus2, serial
except Exception:
    smbus2 = None
    serial = None

# === Імпорт вашого GA-модуля ===
try:
    from greenhouse_ga import run_ga, GAConfig, Targets, Weights, GreenhouseSimulator, EnergyModel
except Exception as e:
    raise RuntimeError("Не знайдено модуль greenhouse_ga. Переконайтесь, що він встановлений у Python path.") from e

# ----------------------------
# Налаштування пінів (приклад BCM)
```

```
# ----------------------------
RELAY_HEAT  = 17
RELAY_FAN   = 27
RELAY_HUMID = 22
RELAY_LIGHT = 23

if HAS_GPIO:
    for pin in (RELAY_HEAT, RELAY_FAN, RELAY_HUMID, RELAY_LIGHT):
        GPIO.setup(pin, GPIO.OUT)
        GPIO.output(pin, False)

# ----------------------------
# Читання сенсорів (поставте свої реалізації, якщо є)
# ----------------------------
def read_sht31() -> (Optional[float], Optional[float]):
    # Температура + Вологість (з SHT31, I2C)
    if smbus2 is None:
        return 24.7, 74.9  # заглушка
    try:
        address = 0x44
        bus = smbus2.SMBus(1)
        bus.write_i2c_block_data(address, 0x2C, [0x06])
        time.sleep(0.5)
        data = bus.read_i2c_block_data(address, 0x00, 6)
        temp_raw = data[0] * 256 + data[1]
        hum_raw  = data[3] * 256 + data[4]
        temperature = -45 + (175 * temp_raw / 65535.0)
        humidity    = 100 * hum_raw / 65535.0
        bus.close()
        return temperature, humidity
    except Exception:
        return None, None

def read_co2() -> Optional[int]:
    # CO2 (MH-Z19B, UART)
    if serial is None:
        return 600  # заглушка
    try:
        ser = serial.Serial("/dev/ttyS1", 9600, timeout=1)
        cmd = bytearray([0xFF,0x01,0x86,0x00,0x00,0x00,0x00,0x00,0x79])
        ser.write(cmd)
        resp = ser.read(9)
        ser.close()
        if len(resp) == 9 and resp[0] == 0xFF and resp[1] == 0x86:
            return (resp[2] << 8) | resp[3]
        return None
    except Exception:
        return None

def read_bh1750() -> Optional[float]:
    # Освітленість (BH1750, I2C)
    if smbus2 is None:
```

```
        return 4500.0  # заглушка
    try:
        addr = 0x23
        bus  = smbus2.SMBus(1)
        bus.write_byte(addr, 0x01)     # power on
        time.sleep(0.02)
        bus.write_byte(addr, 0x10)     # continuous high-res
        time.sleep(0.18)
        data = bus.read_i2c_block_data(addr, 0x00, 2)
        raw  = (data[0] << 8) | data[1]
        lux  = raw / 1.2
        bus.close()
        return lux
    except Exception:
        return None

# ----------------------------
# Глобальний стан (кеш вимірювань, стан акторів)
# ----------------------------
STATE: Dict[str, Any] = {
    "sensors": {
        "temp": None,
        "hum": None,
        "co2": None,
        "lux": None,
        "updated": None,
    },
    "actuators": {
        "heat": False,
        "fan": False,
        "humid": False,
        "light": False,
        "updated": None,
    },
    "ga": {
        "last_fitness": None,
        "last_targets": None,
        "last_actuators": None,
        "updated": None,
    }
}

def set_actuators(heat: bool, fan: bool, humid: bool, light: bool):
    STATE["actuators"].update({
        "heat": bool(heat),
        "fan": bool(fan),
        "humid": bool(humid),
        "light": bool(light),
        "updated": datetime.utcnow().isoformat()
    })
    if HAS_GPIO:
        GPIO.output(RELAY_HEAT,  heat)
```

```
        GPIO.output(RELAY_FAN,   fan)
        GPIO.output(RELAY_HUMID, humid)
        GPIO.output(RELAY_LIGHT, light)

# Початковий стан акторів: викл
set_actuators(False, False, False, False)

# ------------------------------
# Фонове опитування сенсорів
# ------------------------------
def sensor_loop(period_s: float = 3.0):
    while True:
        T, H = read_sht31()
        C = read_co2()
        L = read_bh1750()
        STATE["sensors"].update({
            "temp": round(T, 2) if T is not None else None,
            "hum":  round(H, 2) if H is not None else None,
            "co2":  int(C) if C is not None else None,
            "lux":  round(L, 1) if L is not None else None,
            "updated": datetime.utcnow().isoformat()
        })
        time.sleep(period_s)

threading.Thread(target=sensor_loop, daemon=True).start()

# ------------------------------
# Flask app
# ------------------------------
app = Flask(__name__)

# Головна сторінка (рівень Dashboard)
HTML = """
<!doctype html>
<html lang="uk">
<head>
<meta charset="utf-8">
<title>Greenhouse Dashboard</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
body{font-family:system-ui,-apple-system,Segoe UI,Roboto,Ubuntu,Arial; margin:0;
background:#f7f7f9; color:#111;}
.header{background:#0d6efd;color:#fff;padding:14px 18px;font-weight:600}
.container{max-width:1050px;margin:24px auto;padding:0 16px}
.card{background:#fff;border:1px solid #e5e7eb;border-radius:14px;box-shadow:0 2px 8px
rgba(0,0,0,.04);padding:16px;margin-bottom:16px}
.grid{display:grid;grid-template-columns:repeat(4,1fr);gap:12px}
.tile{background:#f9fafb;border:1px solid #e5e7eb;border-radius:12px;padding:12px}
.tile h3{margin:0 0 6px 0;font-size:14px;color:#374151}
.value{font-size:28px;font-weight:700;margin-top:6px}
.btn{background:#0d6efd;color:#fff;border:none;border-radius:10px;padding:10px
14px;cursor:pointer}
```

```
.btn:disabled{opacity:.6;cursor:not-allowed}
.row{display:flex;gap:12px;align-items:center;flex-wrap:wrap}
.badge{display:inline-block;padding:4px 8px;border-radius:999px;background:#eef2ff;color:#3730a3;font-weight:600;font-size:12px}
table{width:100%;border-collapse:collapse}
td,th{padding:10px;border-bottom:1px solid #e5e7eb;text-align:left}
.switch{display:inline-flex;align-items:center;gap:8px}
.switch input{transform:scale(1.2)}
footer{color:#6b7280;font-size:12px;margin:18px 0}
</style>
</head>
<body>
<div class="header">Інтелектуальна система управління теплицею – Web Dashboard</div>
<div class="container">
  <div class="card">
    <div class="row">
      <div class="badge">Стан сенсорів</div>
      <div id="updated" style="margin-left:auto;color:#6b7280;"></div>
    </div>
    <div class="grid">
      <div class="tile"><h3>Температура</h3><div class="value" id="temp">–</div><div>°C</div></div>
      <div class="tile"><h3>Вологість</h3> <div class="value" id="hum">–</div><div>% RH</div></div>
      <div class="tile"><h3>CO₂</h3> <div class="value" id="co2">–</div><div>ppm</div></div>
      <div class="tile"><h3>Освітленість</h3><div class="value" id="lux">–</div><div>lx</div></div>
    </div>
  </div>

  <div class="card">
    <div class="row">
      <div class="badge">Керування виконавчими пристроями</div>
      <div id="act-updated" style="margin-left:auto;color:#6b7280;"></div>
    </div>
    <div class="grid" style="grid-template-columns:repeat(4,1fr)">
      <div class="tile switch"><input type="checkbox" id="heat"> <label for="heat">Нагрівач</label></div>
      <div class="tile switch"><input type="checkbox" id="fan"> <label for="fan">Вентилятор</label></div>
      <div class="tile switch"><input type="checkbox" id="humid"><label for="humid">Зволоження</label></div>
      <div class="tile switch"><input type="checkbox" id="light"><label for="light">Освітлення</label></div>
    </div>
    <div class="row" style="margin-top:12px;">
      <button class="btn" onclick="sendActuators()">Застосувати</button>
      <button class="btn" onclick="runOptimize()" id="optBtn">Оптимізувати (GA)</button>
      <div id="ga-status" style="margin-left:auto;color:#6b7280;"></div>
    </div>
  </div>
```

```
  <div class="card">
    <div class="row"><div class="badge">Журнал GA</div></div>
    <table>
      <thead><tr><th>Час</th><th>Fitness</th><th>Targets [T,H,C,L]</th><th>Actuators
[heat,fan,humid,light]</th></tr></thead>
      <tbody id="ga-log"></tbody>
    </table>
  </div>

  <footer>© {{year}} Greenhouse Control | Flask Dashboard</footer>
</div>

<script>
async function fetchState(){
  const r = await fetch('/api/state');
  const s = await r.json();

  const se = s.sensors || {};
  document.getElementById('temp').innerText = se.temp ?? '–';
  document.getElementById('hum').innerText  = se.hum ?? '–';
  document.getElementById('co2').innerText  = se.co2 ?? '–';
  document.getElementById('lux').innerText  = se.lux ?? '–';
  document.getElementById('updated').innerText = se.updated ? ('Оновлено: ' + se.updated) : '';

  const ac = s.actuators || {};
  document.getElementById('heat').checked = !!ac.heat;
  document.getElementById('fan').checked  = !!ac.fan;
  document.getElementById('humid').checked= !!ac.humid;
  document.getElementById('light').checked= !!ac.light;
  document.getElementById('act-updated').innerText = ac.updated ? ('Оновлено: ' + ac.updated) : '';

  const ga = s.ga || {};
  if (ga.updated){
    document.getElementById('ga-status').innerText = 'GA: ' + ga.last_fitness + ' (оновлено ' +
ga.updated + ')';
    const row = document.createElement('tr');
    row.innerHTML = `<td>${ga.updated}</td>
                     <td>${ga.last_fitness ?? '–'}</td>
                     <td>${JSON.stringify(ga.last_targets ?? {})}</td>
                     <td>${JSON.stringify(ga.last_actuators ?? {})}</td>`;
    document.getElementById('ga-log').prepend(row);
  }
}

async function sendActuators(){
  const payload = {
    heat:  document.getElementById('heat').checked,
    fan:   document.getElementById('fan').checked,
    humid: document.getElementById('humid').checked,
    light: document.getElementById('light').checked
  };
```

```
  const r = await fetch('/api/actuators', {method:'POST', headers:{'Content-Type':'application/json'},
body:JSON.stringify(payload)});
  await fetchState();
}

async function runOptimize(){
  const btn = document.getElementById('optBtn');
  btn.disabled = true;
  document.getElementById('ga-status').innerText = 'Оптимізація...';
  try{
    const r = await fetch('/api/optimize', {method:'POST'});
    const js = await r.json();
    await fetchState();
    document.getElementById('ga-status').innerText = 'Готово: fitness=' + (js.fitness?.toFixed ?
js.fitness.toFixed(4) : js.fitness);
  } catch(e){
    document.getElementById('ga-status').innerText = 'Помилка оптимізації';
  } finally{
    btn.disabled = false;
  }
}

setInterval(fetchState, 2000);
fetchState();
</script>
</body>
</html>
"""

@app.route("/")
def index():
    return render_template_string(HTML, year=datetime.utcnow().year)

@app.get("/api/state")
def api_state():
    return jsonify(STATE)

@app.post("/api/actuators")
def api_act():
    js = request.get_json(force=True, silent=True) or {}
    set_actuators(bool(js.get("heat")), bool(js.get("fan")), bool(js.get("humid")), bool(js.get("light")))
    return jsonify({"ok": True, "actuators": STATE["actuators"]})

@app.post("/api/optimize")
def api_optimize():
    # Беремо сенсорні дані (із кешу). Якщо чогось нема – ставимо дефолти.
    temp = STATE["sensors"].get("temp") or 24.0
    hum  = STATE["sensors"].get("hum")  or 75.0
    co2  = STATE["sensors"].get("co2")  or 600
    lux  = STATE["sensors"].get("lux")  or 4500.0

    # Зовнішні умови – можна взяти з метео API або задати вручну
```

```
    sim = GreenhouseSimulator(outside_T=20.0, outside_H=65.0, outside_C=430.0, solar_lux=lux)

    cfg = GAConfig(pop_size=50, generations=100)
    res = run_ga(cfg, Targets(), Weights(), sim, EnergyModel())
    b = res.best

    # Застосовуємо рішення
    set_actuators(b.heat > 0.5, b.vent > 0.5, b.humid > 0.5, b.light > 0.5)

    STATE["ga"].update({
        "last_fitness": float(b.fitness),
        "last_targets": {"T": b.T, "H": b.H, "C": b.C, "L": b.L},
        "last_actuators": {"heat": b.heat, "vent": b.vent, "humid": b.humid, "light": b.light},
        "updated": datetime.utcnow().isoformat()
    })
    return jsonify({
        "fitness": float(b.fitness),
        "targets": STATE["ga"]["last_targets"],
        "actuators": STATE["ga"]["last_actuators"]
    })

if __name__ == "__main__":
    try:
        app.run(host="0.0.0.0", port=int(os.getenv("PORT", "8080")), debug=False)
    finally:
        if HAS_GPIO:
            GPIO.cleanup()
```