

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

**Факультет прикладної математики**

**Кафедра системного програмування і спеціалізованих  
комп'ютерних систем**

До захисту допущено:

Завідувач кафедри

\_\_\_\_\_ Віталій РОМАНКЕВИЧ

« \_\_\_\_ » \_\_\_\_\_ 20\_\_ р.

**Дипломний проєкт**

**на здобуття ступеня бакалавра  
за освітньо-професійною програмою  
«Системне програмування та спеціалізовані комп'ютерні системи»  
спеціальності 123 «Комп'ютерна інженерія»**

**на тему: «Інтелектуальна система енергоменеджменту розумного  
будинку з аналізом кліматичних та освітлювальних умов»**

Виконав (-ла):

студент (-ка) IV курсу, групи КВ-12

Кеба Іван Олександрович \_\_\_\_\_

Керівник:

Доцент кафедри СПСіКС к.т.н.

Потапова Катерина Романівна \_\_\_\_\_

Консультант з нормоконтролю:

Доцент кафедри СПСіКС к.т.н.

Клятченко Ярослав Михайлович \_\_\_\_\_

Рецензент:

Доцент кафедри ПМ, к.ф.-м.н.

Вовк Лілія Борисівна \_\_\_\_\_

Засвідчую, що у цьому дипломному  
проєкті немає запозичень з праць інших  
авторів без відповідних посилань.

Студент \_\_\_\_\_

Київ – 2025 року

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

**Факультет прикладної математики  
Кафедра системного програмування і  
спеціалізованих комп'ютерних систем**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування та спеціалізовані комп'ютерні системи»

**ЗАТВЕРДЖУЮ**

**Завідувач кафедри**

\_\_\_\_\_ Віталій РОМАНКЕВИЧ

« \_\_\_ » \_\_\_\_\_ 20\_\_ р.

**ЗАВДАННЯ**

**на дипломний проєкт студенту**

**Кеба Іван Олександрович**

1. Тема проєкту «Інтелектуальна система енергоменеджменту розумного будинку з аналізом кліматичних та освітлювальних умов», керівник проєкту Потапова Катерина Романівна, Доцент кафедри СПСіКС к.т.н., затверджені наказом по університету від «29» травня 2025р. №1808-С

2. Термін подання студентом проєкту 12.06.2025р.

3. Вихідні дані до проєкту див. ТЗ

4. Зміст пояснювальної записки

1) Аналіз існуючих рішень та обґрунтування теми

2) Аналіз обраних технологій та апаратних засобів

3) Тестування, оцінка ефективності та висновки

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

Структурна схема апаратної архітектури системи енергоменеджменту

Схема підключення сенсорів до ESP32

Схема послідовність обробки даних

Схема комунікації ESP32

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормконтроль	Клятченко Я.М., доц. каф. СПіСКС, к.т.н		

7. Дата видачі завдання \_\_\_\_\_

#### Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення літератури за тематики проєкту	09.11.2024	Виконано
2	Розроблення та узгодження технічного завдання	27.11.2024	Виконано
3	Аналіз існуючих рішень	15.12.2024	Виконано
4	Підготовка матеріалів розділів дипломного проєкту	18.02.2025	Виконано
5	Підготовка звіту дипломного проєкту	24.03.2025	Виконано
6	Передзахист дипломного проєкту	27.04.2025	Виконано

Студент

Іван КЕБА

Керівник

Катерина ПОТАПОВА

## АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (69 с., 25 рис., 8 дод.)

Об'єкт розробки – інтелектуальна система енергоменеджменту для розумного будинку з можливістю адаптивного керування кліматичними параметрами на основі аналізу внутрішніх і зовнішніх умов.

Розроблена система забезпечує автоматичне регулювання температури у приміщеннях з урахуванням показників сенсорів температури, вологості, освітленості, а також прогнозу погоди. Реалізовано функціонал прогнозування енергоспоживання за допомогою методів машинного навчання. Передбачено зручний інтерфейс користувача (мобільний або веб), система працює в реальному часі та може бути розгорнута на базі мікроконтролера ESP32.

У процесі розробки:

- проведено аналіз існуючих рішень у сфері керування кліматом і енергоменеджменту в системах розумного будинку;
- визначено вимоги до функціоналу системи;
- розроблено архітектуру системи, включаючи сенсорну підсистему, алгоритм прогнозування та регулювання;
- реалізовано модулі збору даних, обробки, збереження та візуалізації;
- розроблено користувацький інтерфейс для налаштування та моніторингу роботи системи;
- протестовано ефективність прогнозу та керування на модельних даних;
- оцінено енергоефективність запропонованого рішення.

Упровадження цієї системи в побутових умовах дозволить знизити витрати на електроенергію до 30%, покращити комфорт користувачів і забезпечити більш екологічно стійке використання ресурсів.

**Ключові слова:**

**РОЗУМНИЙ ДІМ, ЕНЕРГОМЕНЕДЖМЕНТ, ESP32, ІоТ, СЕНСОРИ, МАШИННЕ НАВЧАННЯ, ПРОГНОЗУВАННЯ, КЛІМАТ-КОНТРОЛЬ.**

## ABSTRACT

The qualification project includes an explanatory note ( 69 pages, 25 figures, 8 appendices)

The object of the development is an intelligent energy management system for a smart home with adaptive control of climate parameters based on the analysis of internal and external conditions.

The developed system provides automatic temperature regulation in premises, taking into account sensor readings for temperature, humidity, illumination, as well as weather forecasts. Functionality for energy consumption prediction using machine learning methods has been implemented. A user-friendly interface (mobile or web) is provided. The system operates in real time and can be deployed on an ESP32 microcontroller.

During the development process:

- an analysis of existing solutions in the field of climate control and energy management for smart home systems was conducted;
- system functionality requirements were defined;
- the system architecture was developed, including the sensor subsystem, prediction and regulation algorithms;
- modules for data collection, processing, storage, and visualization were implemented;
- a user interface for system configuration and monitoring was developed;
- the efficiency of the prediction and control functionality was tested on model data;
- the energy efficiency of the proposed solution was evaluated.

The implementation of this system in residential conditions will reduce electricity costs by up to 30%, improve user comfort, and ensure more environmentally sustainable resource usage.

### **Keywords:**

SMART HOME, ENERGY MANAGEMENT, ESP32, IoT, SENSORS, MACHINE LEARNING, PREDICTION, CLIMATE CONTROL.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	П римітки
	A4	ІАЛЦ.467200.002 ТЗ	Інтелектуальна система енергоменеджменту розумного будинку з аналізом кліматичних та освітлювальних умов	5		
			Технічне завдання			
	A4	ІАЛЦ.467200.003 ТП	Інтелектуальна система енергоменеджменту розумного будинку з аналізом кліматичних та освітлювальних умов	1		
			Відомість технічного проекту			
	A4	ІАЛЦ.467200.004 ПЗ	Інтелектуальна система енергоменеджменту розумного будинку з аналізом кліматичних та освітлювальних умов	69		
			Пояснювальна записка			
	A4	ІАЛЦ.467200.005 Д1	Підключення сенсорів до ESP32	1		
			Принципова схема			

## ***ІАЛЦ.467200.001 ОА***

Змі	Арк.	№ докум.	Підпис	Дата				
Розробив		Кеба І.О.			Інтелектуальна система енергоменеджменту розумного будинку з аналізом кліматичних та освітлювальних умов <b>Опис альбому</b>	Літ.	Аркуш	
Перевірив		Потапова К.Р.					1	
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-12		
Н.		Клятченко Я.М.						
Зав. каф.		Романкевич В.О.						



## ЗМІСТ

1.	НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ.....	2
2.	ПІДСТАВА ДЛЯ РОЗРОБКИ .....	2
3.	МЕТА І ПРИЗНАЧЕННЯ РОБОТИ.....	3
4.	ДЖЕРЕЛА РОЗРОБКИ.....	4
5.	ТЕХНІЧНІ ВИМОГИ.....	4
5.1	Вимоги до програмного продукту, що розробляється .....	4
5.2	Вимоги до апаратного забезпечення .....	4
5.3	Вимоги до програмного та апаратного забезпечення користувача .....	5
6.	ЕТАПИ РОЗРОБКИ .....	5

					ІАЛЦ.467200.002 ТЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		1

## 1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Інтелектуальна система енергоменеджменту для розумного будинку з адаптивним регулюванням температури на основі погодних та освітлювальних умов»

Галузь застосування:

- Системи «розумного будинку» (Smart Home, IoT).
- Енергоменеджмент і енергоефективність житлових та комерційних приміщень.
- Автоматизація клімат-контролю з використанням машинного навчання.
- Дослідницькі та освітні проекти у сфері комп'ютерної інженерії, прикладних математики та енергетики.

## 2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Дипломне завдання зі спеціальності «Комп'ютерна інженерія» (КПІ ім. Ігоря Сікорського), яке передбачає створення інноваційного проекту «розумний дім» із вимірюванням температури, вологості, освітленості та наданням порад через Telegram-бот.

Актуальна необхідність підвищення енергоефективності побутових і офісних приміщень, необхідність підтримання максимально комфортних умов в приміщеннях для комфортної роботи .

Науково-технічна література з управління кліматом у будівлях, методів прогнозування енергоспоживання та адаптивного керування HVAC-системами із застосуванням ML/AI.

Потреба в гнучкому рішенні з відкритим кодом, яке легко модифікувати під різні сценарії, сенсорні конфігурації та умови експлуатації.

					ІАЛЦ.467200.002 ТЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		2

### 3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Розробити інтелектуальну систему енергоменеджменту для «розумного будинку», здатну автоматично регулювати внутрішню температуру з урахуванням внутрішніх та зовнішніх умов, прогнозувати енергоспоживання, покращувати комфорт мешканців.

Призначення роботи:

- Налаштувати мікроконтролер ESP32 для безперервного зчитування показників температури, вологості (DHT22) та освітленості (VEML7700). Реалізувати періодичне оновлення зовнішніх метеоданих за допомогою погодного API для врахування прогнозу погоди.
- Забезпечити збереження зібраної інформації у структурованому вигляді. Дані мають накопичуватися в базі (локально або у хмарному сховищі) з подальшою можливістю аналізу.
- Реалізувати модуль аналізу часових рядів із використанням моделі машинного навчання (зокрема LSTM), яка буде прогнозувати зміни температури, вологості, рівня освітлення або навіть умов енергоспоживання на основі історичних даних та погодного прогнозу.
- На основі аналізу поточних і прогнозованих умов реалізувати програмний алгоритм, що надає користувачу поради — наприклад: *“Завтра очікується зниження температури, рекомендується підвищити обігрів приміщення”*.
- Розробити Telegram-бот, через який користувач зможе переглядати актуальні дані з сенсорів, отримувати рекомендації щодо клімату, а також переглядати графіки змін за останній період.
- Провести тестування системи на основі реальних або змодельованих даних, продемонструвавши точність прогнозування та корисність наданих рекомендацій для підвищення комфорту та можливого зниження споживання енергії.

					ІАЛЦ.467200.002 ТЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		3

#### 4. ДЖЕРЕЛА РОЗРОБКИ

Наукові статті та технічні публікації з управління кліматом у будівлях, прогнозування енергоспоживання, адаптивних алгоритмів (Model Predictive Control, Reinforcement Learning у HVAC тощо).

Приклади та open-source проєкти у сфері Smart Home: документація ESP32, MQTT-брокери, open-source платформи управління кліматом.

API документування OpenWeatherMa; документація API open-meteo.com.

Datasheet сенсорів DHT22, VEML7700; технічні статті щодо їхнього підключення та калібрування.

Бібліотеки та фреймворки: TensorFlow / Keras, scikit-learn, pandas, Flask, APScheduler.

Документація з організації збереження даних: SQLite, Firebase, PostgreSQL.

Методичні матеріали дипломного проєкту: вимоги кафедри, шаблони оформлення.

Підручники з програмування мікроконтролерів (Arduino IDE для ESP32), Python-розробки бекенду, фронтенд-розробки.

#### 5. ТЕХНІЧНІ ВИМОГИ

##### 5.1 Вимоги до програмного продукту

- Функціональні: збір і передачу даних сенсорів кожні 5 хвилин; запит погоди; прогноз споживання; розрахунок комфортної температури; керування через ESP32; інтерфейс для відображення даних, налаштувань; сповіщення Telegram.

- Нефункціональні: обробка помилок при недоступності сенсорів чи мережі; логування подій; стійкість до збоїв; можливість масштабування (додавання сенсорів, перенавчання моделі); безпека токенів та API.

- Продуктивність: оновлення даних у реальному часі, прогноз у межах прийняттого часу.

##### 5.2 Вимоги до апаратного забезпечення

					ІАЛЦ.467200.002 ТЗ	Арк.
						4
Змін.	Арк.	№ докум.	Підпис	Дата		

- ESP32 з Wi-Fi для підключення сенсорів: DHT22 і VEMML7700
- Надійне Wi-Fi-покриття у зоні встановлення ESP32.
- Сервер (ПК) для запуску Flask-дodatка, збереження даних, тренування моделей

### 5.3 Вимоги до ПЗ та АЗ користувача

- Сервер: ОС Windows для розгортання/розробки; Python 3.8+, бібліотеки (Flask, pandas, TensorFlow).
- Клієнт: пристрій із сучасним браузером або мобільний пристрій; встановлений Telegram для отримання сповіщень.
- Мережа: стабільне інтернет-з'єднання або локальна мережа з відкритою можливістю комунікації між ESP32 і сервером.
- Інструменти: Arduino IDE для ESP32; Текстові редактори.

## 6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1	Вивчення літератури та аналіз існуючих рішень	23.11.2024
2	Формулювання та узгодження технічного завдання	07.01.2025
3	Проектування архітектури системи	28.01.2025
4	Налаштування апаратної платформи й початковий збір даних	22.02.2025
5	Розробка та тренування ML-модуля прогнозування	07.03.2025
6	Інтеграція компонентів і розробка інтерфейсу користувача	26.04.2025
7	Тестування системи та оцінка ефективності	05.05.2025
8	Підготовка документації, презентації та підготовка до захисту	29.05.2025

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	A4	ІАЛЦ.467200.004 ПЗ	Інтелектуальна система енергоменеджменту розумного будинку з аналізом кліматичних та освітлювальних умов	69		
	A4	ІАЛЦ.467200.005 Д1	Підключення сенсорів ESP32 Принципова схема	1		
	A4	ІАЛЦ.467200.006 Д2	Послідовність обробки даних Функціональна схема	1		
	A4	ІАЛЦ.467200.007 Д3	Апаратна архітектура системи енергоменеджменту Структурна схема	1		
	A4	ІАЛЦ.467200.008 Д4	Схема комунікації Функціональна схема Диск CD-ROM: текст Текст анотації Архівований код Графічний матеріал			

## ***ІАЛЦ.467200.003 ТП***

Змі	Арк.	№ докум.	Підпис	Дата				
Розробив		Кеба І.О.			Інтелектуальна система енергоменеджменту розумного будинку з аналізом кліматичних та освітлювальних умов <b>Відомість технічного проєкту</b>	Літ.	Аркуш	
Перевірив		Потапова К.Р.					1	
Консульт.						КПІ		
Н. контроль		Клятченко Я.М.				ім. Ігоря Сікорського,		
Зав. каф.		Романкевич В.О.				ФПМ КВ-12		

**Пояснювальна записка**  
**до дипломного проєкту**

на тему: «Інтелектуальна система енергоменеджменту розумного будинку з  
аналізом кліматичних та освітлювальних умов»

Київ – 2025

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		1

## ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ.....	5
ВСТУП.....	6
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ.....	7
1.1 Огляд комерційних рішень Smart Home з клімат-контролем .....	7
1.2 Наукові підходи до адаптивного керування кліматом .....	8
1.3 Обмеження існуючих підходів .....	9
1.4 Висновки та обґрунтування вибору теми.....	11
2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ.....	12
2.1. Аналіз обраних технологій та апаратних засобів .....	12
2.1.1 Загальна концепція модульності.....	12
2.1.2 Взаємодія компонентів і потоки даних .....	14
2.1.3 Переваги та обґрунтування вибору модульної архітектури .....	15
2.2. Апаратна частина і підключення сенсорів .....	17
2.2.1 Вибір мікроконтролера: ESP32 .....	17
2.2.2 Сенсори внутрішнього середовища .....	18
2.2.3 Живлення і стабілізація напруги .....	19
2.2.4 Мережеве з'єднання .....	20
2.2.5 Відсутність реле та можливі напрямки еволюції .....	21
2.3. Прошивка ESP32 .....	22
2.4. Серверна частина на Flask .....	23
2.4.1 Організація проєкту .....	24
2.4.2 Ініціалізація Flask і APScheduler .....	24
2.4.3 Маршрути Flask .....	25
2.4.4 Модулі прогнозу погоди і ML-аналіз .....	26
2.4.5 Обробка винятків та логування .....	27
2.4.6 Перевірка та ініціалізація файлів .....	27
2.4.7 Налаштування APScheduler та часові зони .....	28
2.4.8 Конфігурація Telegram-бота на сервері .....	29
2.4.9 Обробка багатопоточності та конкуренції доступу до файлів.....	30

Змін.	Арк.	№ докум.	Підпис	Дата

2.5. Модуль прогнозу погоди .....	31
2.5.1 Вибір джерела метеоданих .....	31
2.5.2 Функція отримання прогнозу на наступну годину .....	32
2.5.3 Логування прогнозу .....	33
2.5.4 Налаштування періодичності .....	34
2.5.5 Обробка обмежень і відмов .....	35
2.6. ML-модуль та алгоритм розрахунку комфортної температури .....	35
2.6.1 Збір і підготовка даних .....	36
2.6.2 Формування даних для LSTM .....	37
2.6.3 Завантаження або тренування моделі .....	38
2.6.4 Прогноз внутрішньої температури .....	39
2.6.5 Алгоритм розрахунку комфортної температури .....	39
2.6.6 Оновлення файлу комфортної температури .....	41
2.6.7 Логування і обробка виключень .....	41
2.6.8 Розширюваність ML-модуля .....	42
2.7. Інтеграція компонентів і періодичний запуск .....	43
2.7.1 Налаштування APScheduler у Flask .....	43
2.7.2 Послідовність роботи .....	44
2.7.3 Обробка виключень у періодичних завданнях .....	46
2.7.4 Налаштування інтервалів та моніторинг .....	46
2.7.5 Мультизапуск пристроїв .....	47
2.7.6 Узгодження з прошивкою ESP32.....	47
2.8. Інтерфейс користувача та діагностика .....	47
2.8.1 Telegram-сповіщення як основний канал взаємодії .....	48
2.8.2 Перегляд файлів і побудова графіків .....	49
2.8.3 Логування подій .....	50
3. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ .....	52
3.1. Тестування окремих компонентів .....	52
3.1.1 Тестування ESP32 .....	52
3.1.2 Тестування серверної частини (Flask) .....	55

3.1.3 Тестування модуля прогнозу погоди.....	56
3.1.4 Тестування ML-модуля .....	58
3.1.5 Тестування алгоритму розрахунку комфортної температури .....	59
3.1.6 Тестування Telegram-бота .....	60
3.2. Інтеграційні тести .....	61
3.3. Оцінка потенційної ефективності .....	63
ВИСНОВКИ .....	66
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....	68
ДОДАТКИ	

## ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ESP32 — мікроконтролер із Wi-Fi модулем.

DHT22 — датчик температури й вологості.

VEML7700 — датчик освітленості.

Wi-Fi — бездротове з'єднання між ESP32 і сервером.

HTTP — протокол обміну даними між ESP32 і Flask-сервером.

CSV — формат збереження даних сенсорів у файл

API — інтерфейс для запиту зовнішніх даних погоди

ML (машинне навчання) — загальний термін для модуля прогнозування споживання/температури.

LSTM — модель довготривалої пам'яті Long Short-Term Memory для прогнозу наступної температури.

Flask — фреймворк Python для організації серверної частини

APScheduler — бібліотека для періодичного запуску `save_next_hour_forecast()` і `run_analysis()`.

Telegram-бот — канал сповіщення користувача; містить `botToken`, `chatID`, API Telegram.

IoT — зазначення, що система є прикладом Інтернету речей.

CLI — інтерфейс командного рядка.

Серійний монітор — у контексті налагодження прошивки ESP32.

`datetime` — поле з датою й часом у CSV-файлах: формат "YYYY-MM-DD HH:MM:SS".

`SEQ_LEN` — довжина послідовності для LSTM (наприклад, 5 записів).

`MinMaxScaler` — метод нормалізації даних перед LSTM.

MSE, RMSE — метрики якості прогнозу (Mean Squared Error, Root MSE).

`timeout` — таймаут мережевих запитів.

## ВСТУП

У теперішніх умовах постійно зростаючого навантаження на енергетичні ресурси та не стабільного клімату особливої актуальності набуває рішення для оптимізації регулювання температури у побутовому середовищі. Розумні системи керування кліматом у приміщеннях мають змогу не лише підвищити комфорт мешканців, але й мінімізувати витрати енергії. Проте багато готових комерційних продуктів мають обмежену адаптивність: зазвичай вони працюють за заздалегідь заданими алгоритмами без гнучкого врахування історії споживання чи прогнозу зовнішніх умов.

Метою даного дослідження є створення розумної системи управління енергоспоживанням у домашньому середовищі, яка, працюючи на обмеженому апаратному ресурсі (ESP32, базові сенсори, без використання реле та повноцінної бази даних), здатна самостійно оцінювати поточний мікроклімат, враховувати прогнози погоди та застосовувати алгоритми машинного навчання для визначення оптимальних температурних умов. Дані зберігаються у вигляді локальних файлів, а обмін з користувачем здійснюється через повідомлення в Telegram або перегляд логів вручну. Такий підхід дозволяє реалізувати працездатний прототип без складної інфраструктури, демонструючи базові можливості адаптивного управління з елементами самонавчання.

У вступі обґрунтовано актуальність теми, яка полягає в необхідності підвищення енергоефективності в житлових і малих комерційних приміщеннях. Завдяки аналізу накопичених даних про мікроклімат (температура, вологість, рівень освітлення) у поєднанні з прогнозом погодних умов можливо заздалегідь оцінювати потребу в обігріві або охолодженні. Використання методів машинного навчання, зокрема рекурентних нейронних мереж (LSTM), дозволяє передбачати зміни температури всередині приміщення з високою точністю. Поєднання такого прогнозування з логікою, заснованою на простих правилах, забезпечує ефективне визначення комфортного температурного режиму навіть при роботі на пристроях з обмеженими обчислювальними ресурсами.

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		4

Завдання роботи включають:

- огляд існуючих рішень із керування кліматом у «розумному будинку» та визначення обмежень комерційних систем щодо адаптивності та самонавчання;
- опис апаратної конфігурації: ESP32 з підключеними датчиками температури, вологості (DHT22 або BME280) та освітленості (VEML7700);
- розробка серверної частини на Flask, яка приймає дані, зберігає їх у файли CSV/TXT, запускає періодичні завдання для збору прогнозу погоди та аналізу даних;
- побудова модуля машинного навчання для прогнозу внутрішньої температури на основі історії та зовнішніх факторів;
- реалізація алгоритму розрахунку комфортної температури, її передача на ESP32 і надсилання рекомендації користувачу через Telegram;
- тестування прототипу в різних сценаріях (зміна зовнішніх умов, втрати зв'язку, обмежена кількість даних) та оцінка потенційної енергоефективності.

Оскільки в поточній версії прототипу не передбачено реле чи інших виконавчих пристроїв, система виконує лише інформативну функцію — надсилає користувачу рекомендації щодо дій, наприклад, вручну активувати чи деактивувати опалювальні або охолоджувальні прилади. Незважаючи на це, експериментальні результати підтверджують ефективність обраного підходу та свідчать про потенціал подальшого розвитку системи з можливістю автоматичного керування. Використання файлового зберігання даних замість розгортання повноцінної СУБД значно спрощує реалізацію, при цьому даючи змогу накопичувати історичну інформацію для аналітики та донавчання моделей.

Структура роботи охоплює такі розділи: аналіз аналогічних рішень та обґрунтування обраної концепції; побудову архітектури системи та характеристику апаратної частини; опис реалізації програмного забезпечення як на стороні ESP32, так і на сервері; процес збору та підготовки даних; побудову

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		5

модуля машинного навчання та алгоритмів обчислення комфортної температури; організацію періодичних завдань і механізму Telegram-сповіщень; демонстрацію отриманих результатів та оцінку впливу системи на комфорт і енергоспоживання; підсумкові висновки з аналізом обмежень і рекомендаціями щодо подальшого вдосконалення.

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		6

## 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

### 1.1. Огляд комерційних рішень Smart Home з клімат-контролем

Більшість сучасних комерційних рішень у сфері «розумного дому» орієнтовані на управління освітленням, безпековими системами та базовою автоматизацією клімату, що зазвичай реалізується через фіксовані розклади або заздалегідь налаштовані сценарії. Наприклад, екосистема LIVOLO забезпечує дистанційне керування освітленням і температурними параметрами, однак її логіка базується переважно на статичних правилах без урахування історичних даних або метеопрогнозів.

На відміну від таких комерційних систем, запропонований проєкт реалізує адаптивний підхід до керування кліматом, що базується на зборі актуальних даних із сенсорів та прогнозуванні внутрішньої температури за допомогою методів машинного навчання. Система не обмежується фіксованими сценаріями: вона динамічно розраховує комфортну температуру з урахуванням зовнішніх умов і поточних показників, надаючи користувачу персоналізовані рекомендації.



Рис. 1.1 – Терморегулятор сенсорний VL-C7FCQ1-2WP з датчиком температури повітря

Система Domos також пропонує інструменти автоматизації для опалення й кондиціонування із можливістю ручного регулювання через мобільний додаток,

але позбавлена функцій прогнозування навантаження або адаптації керування на основі машинного навчання. Завдяки використанню прогнозу температури та адаптивного розрахунку комфортного клімату система формує рекомендації, що враховують як поточні, так і очікувані умови.

Модуль машинного навчання дозволяє прогнозувати зміну температури на основі історичних даних, а також адаптувати рекомендації відповідно до різних ситуацій, наприклад, різкої зміни погоди. Цей підхід дозволяє підвищити ефективність енергоспоживання та забезпечити стабільніші умови без необхідності постійного втручання користувача. Таким чином, навіть за відсутності повноцінного інтерфейсу автоматизації, система демонструє крок у напрямку інтелектуального управління, чого бракує більшості наявних комерційних рішень



Рис. 1.2 – Терморегулятор Domos MK72-GB

Рішення Grenton, яке розповсюджується через компанію SaniWell, використовує сенсори зовнішнього середовища (температура, освітленість) для покращення мікроклімату і базового енергомоніторингу, однак не передбачає аналізу історичних трендів і не має вбудованих алгоритмів прогнозування споживання.

Отже, попри широке впровадження технологій автоматизації, більшість доступних на ринку платформ оперують передусім статичними підходами, не інтегруючи повноцінні механізми самонавчання, персоналізації або адаптації до змін у поведінці користувачів чи зовнішніх умов.

## 1.2. Наукові підходи до адаптивного керування кліматом

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		8

У сучасних дослідженнях з енергоменеджменту приміщень широко впроваджуються підходи на основі Model Predictive Control (MPC), де математична модель теплових процесів будівлі в поєднанні з прогнозом погодних умов дозволяє реалізувати оптимальне управління системами HVAC з мінімізацією енергозатрат. За результатами експериментів, такі стратегії забезпечують зменшення споживання без шкоди для комфорту мешканців, однак вимагають детальної інформації про характеристики будівлі та наявності ресурсів для обчислення оптимальних дій у реальному часі (джерело: arxiv.org).

Альтернативою MPC виступає підхід із використанням методів навчання з підкріпленням (reinforcement learning, RL), де агент самостійно формує політику керування на основі оцінки досягнутого комфорту і витрат енергії. Попри потенційну ефективність, цей метод потребує або великої кількості історичних даних, або симуляційного середовища для попереднього навчання, що обмежує його застосування в системах із обмеженими обчислювальними ресурсами.

Ще одним напрямом є управління з орієнтацією на користувача (occupant-centric control), коли рішення приймаються з урахуванням присутності людей у приміщенні, їхніх індивідуальних уподобань і поведінкових шаблонів. Такі стратегії демонструють значну ефективність у скороченні енергоспоживання, однак вимагають інтеграції додаткових сенсорів і складнішої логіки аналізу.

Таким чином, хоча поєднання MPC, RL і користувацько-орієнтованих підходів забезпечує високу ефективність у керуванні мікрокліматом, практична реалізація таких систем потребує потужної інфраструктури, широкої сенсорної мережі та високої обчислювальної здатності, що не завжди можливо в умовах обмежених апаратних ресурсів чи освітніх проєктів.

### 1.3. Обмеження існуючих підходів

- Інфраструктурні й фінансові бар'єри: професійні BMS та хмарні сервіси потребують інвестицій у сервери, ліцензії, налаштування моделей будівлі та інтеграцію з існуючими HVAC, що недоцільно для малих об'єктів або студентських проєктів.

- Обмежені ресурси апаратної платформи: контролери ESP32 та базові сенсори (DHT22, VEMML7700) не призначені для виконання важких обчислень на місці; відсутність реле та інших факторів означає, що система може лише надавати рекомендації для ручного керування обладнанням.

- Відсутність СУБД: замість складної бази даних використовується збереження в CSV, що спрощує реалізацію, але накладає обмеження на обсяг і безпеку збережених даних.

- Надійність мережі: простий Wi-Fi зв'язок ESP32 ↔ сервер потребує обробки випадків втрати підключення шляхом кешування та повторних спроб, але не вимагає складних мережевих архітектур.

- Для зручності користувача взаємодія із системою реалізована через Telegram-бота, що дозволяє оперативно отримувати сповіщення про стан системи, рекомендації щодо дій та перегляд основної інформації без потреби у розгортанні окремого мобільного або веб-додатку.

- Навчальна цінність: дипломний проєкт має продемонструвати архітектуру адаптивної системи із прогнозом і правилами управління, але без розгортання складної корпоративної платформи.

Виходячи з цього, формується потреба в прототипі, який:

- використовує ESP32 із датчиками температури, вологості, освітленості;
- зберігає дані в файловій системі (CSV/TXT) без СУБД;
- отримує прогнози погоди через публічні API;
- виконує прогнозування внутрішньої температури на сервері з Python (LSTM);
- розраховує комфортну температуру за поєднанням правил і прогнозу;
- передає рекомендації на ESP32 і надсилає повідомлення через Telegram;
- не здійснює прямого керування, але демонструє потенціал енергоефективності та комфорту;

- слугує основою для подальшого розширення (додавання реле, СУБД, UI).

#### 1.4. Висновки та обґрунтування вибору теми

Аналіз наявних комерційних систем засвідчує, що хоча керування мікрокліматом вже активно впроваджується, воно здебільшого реалізоване у вигляді фіксованих сценаріїв без динамічної адаптації до змін умов або індивідуальної історії.

Натомість академічні підходи — такі як MPC, RL чи системи, орієнтовані на присутність користувачів — демонструють високу ефективність, але вимагають складної технічної бази, численних сенсорів і обчислювальних потужностей. У межах цього дипломного проєкту основний акцент зроблено на демонстрацію того, що навіть з мінімальними апаратними ресурсами (ESP32, базові сенсори) можливо реалізувати концепцію адаптивного енергоменеджменту.

Зокрема, система здійснює збір параметрів середовища, прогнозує зміну температури з використанням моделей машинного навчання, обчислює комфортну температуру та надсилає користувачу рекомендації.

Відсутність актуаторів та повноцінної бази даних не обмежує ідею: навіть у режимі ручного втручання користувача можна досягти зниження енергоспоживання та стабілізації мікроклімату. Формат збереження даних у вигляді файлів є простим у реалізації та відповідає технічним обмеженням. Таким чином, обрана тема є актуальною та перспективною, оскільки дозволяє показати ефективність алгоритмічного підходу в умовах обмежених ресурсів без запозичення готових рішень.

## 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

### 2.1. Аналіз обраних технологій та апаратних засобів

Була розроблена модульна архітектура, яка дозволяє чітко розділити обов'язки кожного блоку системи, забезпечити простоту тестування і подальшого розвитку.

#### 2.1.1. Загальна концепція модульності

Система реалізована з урахуванням принципів модульності, що передбачає логічне розділення на незалежні компоненти з чітко визначеними обов'язками. Такий підхід спрощує розробку, налагодження та розширення проєкту як на рівні мікроконтролера, так і серверної частини.

Комунікація між ESP32 та сервером побудована за допомогою стандартних HTTP-запитів. На стороні мікроконтролера відбувається зчитування даних із сенсорів, формування JSON-пакету з поточними значеннями температури, вологості та освітленості, після чого дані надсилаються на сервер у вигляді POST-запиту. У відповідь сервер формує статус-код, який ESP32 обробляє та виводить у логах, що забезпечує зворотний зв'язок і можливість діагностики з'єднання.

Серверна частина приймає ці дані за маршрутом /data, перетворює на числові значення та додає у файл sensor\_data.csv з автоматичною часовою міткою. Кожен запис зберігається у вигляді окремого рядка, що дозволяє будувати часові ряди та здійснювати машинний аналіз у подальшому.

Періодичні задачі, зокрема прогнозування зовнішньої температури та розрахунок комфортного клімату, виконуються незалежно від мікроконтролера завдяки планувальнику завдань APScheduler. Окремо кожні 30 хвилин запускається функція, що звертається до погодного API, зберігає прогноз у файл forecast\_hourly.csv, після чого ініціюється виконання модуля run\_analysis(), який оновлює модель, робить короткостроковий прогноз внутрішньої температури та обчислює рекомендовану комфортну температуру. Результат записується в окремий файл comfort\_temp.txt.

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		12

З боку ESP32, у задані інтервали виконується запит GET на сервер для отримання останньої розрахованої комфортної температури. Отримане значення використовується для інформування користувача через Telegram-бота, який надсилає коротке повідомлення з поточними показниками та рекомендацією.

Поділ проєкту на окремі логічні частини забезпечує низку важливих переваг:

- Розмежування відповідальності. Кожен компонент виконує чітко визначене завдання — обробка даних сенсорів, комунікація з API, аналіз, логування або інтерфейс із користувачем. Це дозволяє розвивати та оновлювати частини системи незалежно.
- Гнучкість. У майбутньому можна безболісно додати нові типи сенсорів, підключити базу даних замість текстових файлів, реалізувати веб-інтерфейс або навіть перенести частину обчислень на хмарну платформу.
- Простота обслуговування. Зберігання даних у текстових файлах полегшує налагодження та тестування, зменшуючи кількість зовнішніх залежностей. Це особливо важливо на етапі демонстрації або під час роботи у ресурсно обмеженому середовищі (наприклад, Raspberry Pi).
- Низький рівень взаємозалежності. Серверна частина, яка виконує складні обчислення та аналіз, повністю ізольована від мікроконтролера. Це дозволяє залишити прошивку ESP32 простою, стабільною і легкою для оновлення.
- Логування та діагностика. Усі ключові етапи роботи системи супроводжуються виводом у консоль або в лог, що значно спрощує виявлення проблем або помилок у роботі.
- Використання стандартних протоколів та форматів. Взаємодія між компонентами побудована на основі загальноприйнятих рішень — HTTP, JSON, CSV — що забезпечує сумісність із зовнішніми системами, простоту інтеграції та мінімальний поріг входу для інших розробників.

Завдяки такій архітектурі система є не лише функціонально повною, а й готовою до подальшого масштабування.

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		13

### 2.1.2. Взаємодія компонентів і потоки даних

Система реалізована з урахуванням принципів модульності, що передбачає логічне розділення на незалежні компоненти з чітко визначеними обов'язками. Такий підхід спрощує розробку, налагодження та розширення проєкту як на рівні мікроконтролера, так і серверної частини.

Комунікація між ESP32 та сервером побудована за допомогою стандартних HTTP-запитів. На стороні мікроконтролера відбувається зчитування даних із сенсорів, формування JSON-паketу з поточними значеннями температури, вологості та освітленості, після чого дані надсилаються єва лучша на сервер у вигляді POST-запиту. У відповідь сервер формує статус-код, який ESP32 обробляє та виводить у логах, що забезпечує зворотний зв'язок і можливість діагностики з'єднання.

Серверна частина приймає ці дані за маршрутом /data, перетворює на числові значення та додає у файл sensor\_data.csv з автоматичною часовою міткою. Кожен запис зберігається у вигляді окремого рядка, що дозволяє будувати часові ряди та здійснювати машинний аналіз у подальшому.

Періодичні задачі, зокрема прогнозування зовнішньої температури та розрахунок комфортного клімату, виконуються незалежно від мікроконтролера завдяки планувальнику завдань APScheduler. Окремо кожні 30 хвилин запускається функція, що звертається до погодного API, зберігає прогноз у файл forecast\_hourly.csv, після чого ініціюється виконання модуля run\_analysis(), який оновлює модель, робить короткостроковий прогноз внутрішньої температури та обчислює рекомендовану комфортну температуру. Результат записується в окремий файл comfort\_temp.txt.

З боку ESP32, у задані інтервали виконується запит GET на сервер для отримання останньої розрахованої комфортної температури. Отримане значення використовується для інформування користувача через Telegram-бота, який надсилає коротке повідомлення з поточними показниками та рекомендацією. Поділ проєкту на окремі логічні частини забезпечує низку важливих переваг:

- Розмежування відповідальності. Кожен компонент виконує чітко визначене завдання — обробка даних сенсорів, комунікація з API, аналіз, логування або інтерфейс із користувачем. Це дозволяє розвивати та оновлювати частини системи незалежно.
- Гнучкість. У майбутньому можна безболісно додати нові типи сенсорів, підключити базу даних замість текстових файлів, реалізувати веб-інтерфейс або навіть перенести частину обчислень на хмарну платформу.
- Простота обслуговування. Зберігання даних у текстових файлах полегшує налагодження та тестування, зменшуючи кількість зовнішніх залежностей. Це особливо важливо на етапі демонстрації або під час роботи у ресурсно обмеженому середовищі (наприклад, Raspberry Pi).
- Низький рівень взаємозалежності. Серверна частина, яка виконує складні обчислення та аналіз, повністю ізольована від мікроконтролера. Це дозволяє залишити прошивку ESP32 простою, стабільною і легкою для оновлення.
- Логування та діагностика. Усі ключові етапи роботи системи супроводжуються виводом у консоль або в лог, що значно спрощує виявлення проблем або помилок у роботі.
- Використання стандартних протоколів та форматів. Взаємодія між компонентами побудована на основі загальноприйнятих рішень — HTTP, JSON, CSV — що забезпечує сумісність із зовнішніми системами, простоту інтеграції та мінімальний поріг входу для інших розробників.

Завдяки такій архітектурі система є не лише функціонально повною, а й готовою до подальшого масштабування.

### 2.1.3. Переваги та обґрунтування вибору модульної архітектури

Проєкт реалізовано за модульною архітектурною схемою, що передбачає поділ системи на окремі компоненти з чітко визначеною функціональністю. Такий підхід дозволяє ізолювати задачі прошивки, серверної логіки, обробки

прогнозів, аналітики та взаємодії з користувачем, що значно спрощує розробку, тестування та супровід окремих частин системи.

Однією з ключових переваг модульного підходу є гнучкість масштабування. За потреби до системи можна легко додати нові сенсори, підключити альтернативні джерела даних, замінити або розширити модель машинного навчання, а також інтегрувати базу даних замість файлового сховища. Це відкриває можливості для поступової еволюції проєкту без необхідності повної перебудови архітектури.

Важливою особливістю є простота початкового налаштування. Усі дані зберігаються у форматах CSV або TXT, що дозволяє уникнути залежності від сторонніх серверів баз даних. Це значно полегшує розгортання системи в умовах обмеженого середовища, наприклад, на одноплатному комп'ютері або під час демонстрації прототипу.

Модульність також забезпечує незалежність окремих частин системи. Мікроконтролер ESP32 відповідає виключно за зчитування та передачу даних, уникаючи обчислювального навантаження, пов'язаного з аналітикою. Водночас серверна частина, реалізована на Python, може виконувати ресурсомісткі задачі аналізу на більш продуктивному обладнанні — зокрема, на ПК або Raspberry Pi. Такий розподіл дозволяє досягти оптимального балансу між ефективністю й стабільністю.

Додаткову перевагу становить прозоре логування процесів. Усі ключові етапи — зчитування даних, формування прогнозу, збереження результатів — супроводжуються виводом у консоль, що полегшує налагодження та пришвидшує виявлення потенційних збоїв.

Нарешті, взаємодія між модулями організована за допомогою поширених інтерфейсів — HTTP-запитів (POST/GET), обміну JSON-об'єктами, роботи з текстовими файлами. Це дозволяє зберегти максимальну сумісність із іншими платформами та мінімізувати вхідний бар'єр для сторонніх розробників або потенційних інтеграторів.

## 2.2. Апаратна частина і підключення сенсорів

У цьому розділі описано вибір апаратних компонентів, підключення сенсорів до ESP32, схему живлення та забезпечення стабільності сигналів. Наведено досвід налаштування, особливості калібрування сенсорів і вжиті заходи для забезпечення надійної роботи системи.

### 2.2.1. Вибір мікроконтролера: ESP32

ESP32 має вбудований Wi-Fi, достатню кількість GPIO-портів, апаратне підтримання I2C, достатню оперативну пам'ять для виконання прошивки. Він широко розповсюджений, має активне співтовариство та бібліотеки для Arduino IDE/PlatformIO. Я обрав популярну плату ESP32 DevKit (типу ESP32-WROOM), яка має стабільне живлення через USB і виводи GPIO доступні для підключення сенсорів. В середовищі Arduino IDE додано необхідні налаштування плати ESP32 для зручності відлагодження. Забезпечено актуальність бібліотек DHT та Adafruit\_VEMML7700.

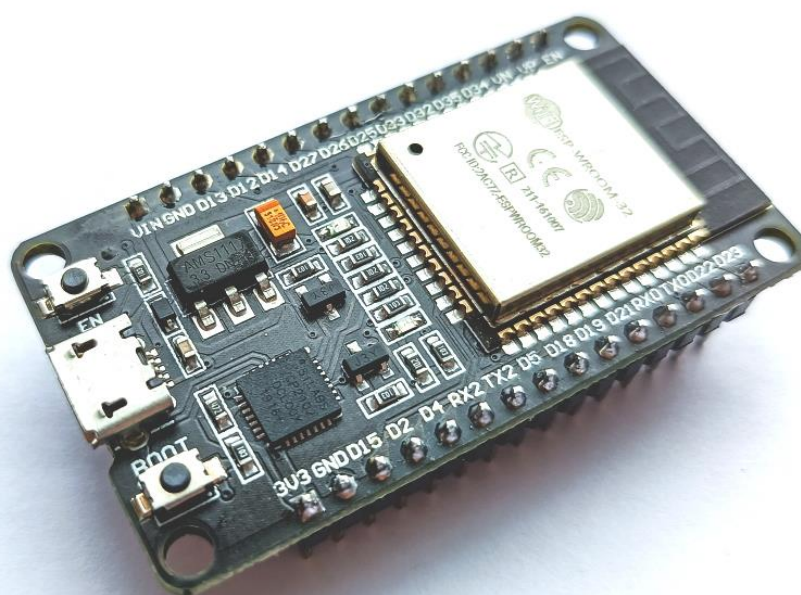


Рис. 2.1 – Модуль ESP32 WROOM-32

## 2.2.2. Сенсори внутрішнього середовища

### DHT22 (BME280 як краща альтернатива)

У реалізованій системі застосовується цифровий датчик температури й вологості DHT22, який оснащений вбудованим модулем обробки даних та передає результати вимірювання через єдиний цифровий пін. Для підключення до мікроконтролера ESP32 було використано живлення від 3.3V, що забезпечує сумісність і безпеку для контролера, хоча датчик також підтримує роботу від 5V.

Контакт GND під'єднано до спільної землі, а сигнальний вихід DATA з'єднано з цифровим входом (GPIO4). Для стабільної роботи між контактами DATA і VCC встановлено підтягуючий резистор опором у межах 4.7–10 кОм, як рекомендовано в технічній документації.

Надійність даних перевірялася за допомогою кількох серій вимірювань у контрольованих умовах, зіставляючи показники з іншими термометрами. З урахуванням технічних особливостей DHT22, а саме обмеження частоти опитування (не частіше ніж раз на 2 секунди), в системі передбачено інтервал зчитування раз на 60 секунд. Цей режим оптимальний для нашого сценарію, де важливе не миттєве реагування, а стабільне відстеження змін температури та вологості з подальшим аналізом. Такий підхід дозволяє забезпечити ефективну роботу з мінімальним навантаженням на обчислювальні ресурси ESP32.



Рис. 2.2 – Модуль DHT22

### VEML7700 (Adafruit\_VEMML7700) для освітленості

Для вимірювання рівня освітленості в системі використовується цифровий сенсор VEML7700, який працює за інтерфейсом I2C і повертає значення

Змін.	Арк.	№ докум.	Підпис	Дата

освітленості у люксах. Підключення до мікроконтролера ESP32 реалізовано через стандартні виводи I2C: лінія SDA під'єднана до GPIO21, а SCL – до GPIO22 згідно зі схемою типового модуля ESP32 DevKit. Сенсор живиться від напруги 3.3V, а загальна земля поєднана з GND ESP32.

Для спрощення взаємодії з пристроєм було використано бібліотеку Adafruit\_VEML7700, яка забезпечує швидку ініціалізацію та стабільне зчитування даних. Роботу сенсора перевірено шляхом створення змін освітленості – наприклад, за допомогою лампи чи часткового перекриття світла. Показники змінювалися відповідно до змін умов, що свідчить про коректне функціонування сенсора та адекватну реакцію на зовнішнє освітлення.



Рис. 2.3 – Модуль VEML7700

### 2.2.3. Живлення і стабілізація напруги

Живлення для ESP32 у проєкті здійснювалося через стандартний USB-адаптер з вихідною напругою 5V. Оскільки плата має вбудований стабілізатор, це дозволяє отримати необхідні 3.3V для живлення самої мікросхеми. У випадку автономного використання без підключення до комп'ютера, замість USB можна застосувати окремий зовнішній адаптер на 5V.

Усі сенсори також були підключені до лінії 3.3V, що генерується ESP32, і попередньо було перевірено, що споживаний струм не перевищує можливостей вбудованого стабілізатора. Для забезпечення надійної роботи усіх елементів окрема увага приділялася якості підключення загальної землі між ESP32 і датчиками. Хоча ESP32 підтримує роботу з файловими системами на основі SPIFFS або SD-карт, у цій реалізації дані зберігаються на сервері. Проте, якщо

виникне потреба у локальному кешуванні, для тимчасового збереження показників можна задіяти SPIFFS.

#### 2.2.4. Мережеве з'єднання

Підключення ESP32 до мережі Wi-Fi реалізоване через збереження в прошивці імені мережі (SSID) та пароля. Під час ініціалізації виконується підключення за допомогою функції WiFi.begin(), а далі — активне очікування підключення в циклі з таймером. Якщо протягом заданого інтервалу підключення не встановлюється, пристрій або повторює спробу, або фіксує помилку в лог.

Така перевірка виконується з певною періодичністю, наприклад, раз на хвилину. Для цілей цього прототипу серверна частина системи розгортається у локальній мережі, з фіксованою IP-адресою — наприклад, на ПК чи Raspberry Pi. Це дозволяє ESP32 стабільно надсилати запити без необхідності зовнішнього доступу. Якщо б виникла потреба в керуванні з будь-якого місця, можливо було б реалізувати доступ через VPN або шляхом пробросу портів, але для дипломної реалізації таких заходів не потрібно. У прошивці реалізована базова перевірка доступності мережі: перед кожною передачею даних виконується перевірка статусу з'єднання.

У випадку його втрати пристрій намагається самостійно перепідключитися, а виміряні значення тимчасово зберігаються у змінних або, за потреби, у файловій системі SPIFFS, щоб надіслати їх пізніше. Щодо безпеки — для локальної мережі в межах прототипу достатньо обмеженого доступу до IP-адреси сервера. Для захисту зовнішніх сервісів, таких як Telegram-бот, URL-адреси та токени були винесені у конфігураційні файли й не містяться безпосередньо в коді, що дозволяє уникнути випадкового розголошення важливої інформації. Це особливо важливо для реального розгортання системи в майбутньому.

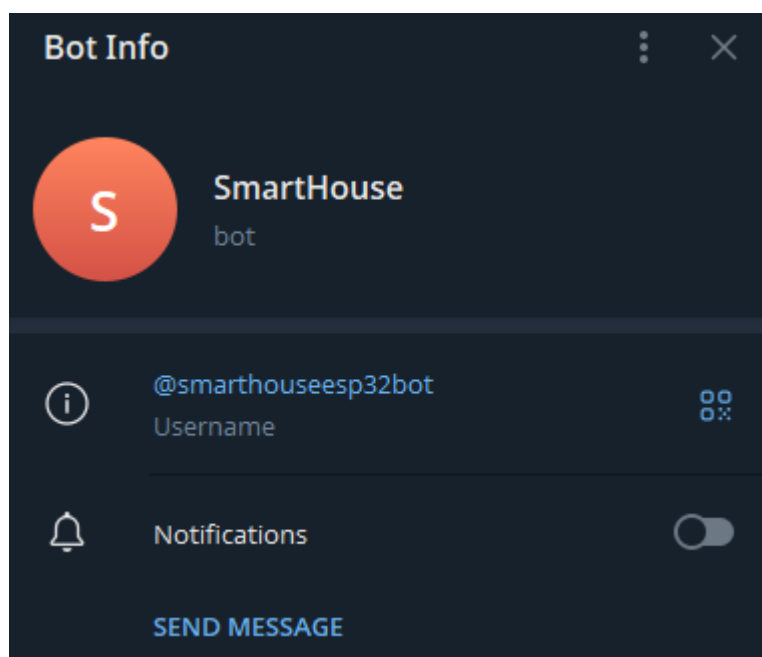


Рис. 2.4 – Зовнішній вигляд телеграм бота

#### 2.2.5. Відсутність реле та можливі напрямки еволюції

На поточному етапі мій прототип не містить виконавчих пристроїв, зокрема реле, тому система виконує лише моніторинг умов і надсилає користувачеві рекомендації щодо оптимальної температури. Такий підхід дозволив сконцентрувати зусилля на зборі даних, їхньому аналізі, прогнозуванні та визначенні комфортного температурного режиму, не ускладнюючи систему питаннями керування електричними навантаженнями та забезпеченням апаратної безпеки.

Проте в майбутньому систему можна доповнити автоматичним управлінням кліматичними пристроями — наприклад, обігрівачем або кондиціонером. Для цього передбачається використання реле або спеціального модуля керування силовими колами. Важливо підібрати реле, яке витримує робочу напругу й струм конкретного пристрою, а також має захист від імпульсних перевантажень. З технічного боку реле можна підключити до ESP32 через транзисторний ключ або готовий модуль із гальванічною розв'язкою.

У прошивку потрібно додати логіку порівняння поточної температури з розрахованою комфортною (`comfort_temp`) і відповідного керування реле.

Щоб уникнути частих перемикань, доцільно реалізувати гістерезис або затримки вмикання/вимикання. На серверній частині можна задавати ці порогові значення гнучко — наприклад, у вигляді налаштувань користувача.

### 2.3. Прошивка ESP32

Прошивка для мікроконтролера ESP32 реалізована як набір функціональних блоків, що забезпечують підключення до мережі Wi-Fi, зчитування показників із сенсорів, передачу даних на сервер та отримання рекомендацій від аналітичного модуля.

На етапі ініціалізації оголошуються параметри мережі (`const char* ssid`, `const char* password`), адреси для HTTP-запитів до серверної частини (`serverURL`, `comfortEndpoint`), а також ідентифікаційні дані Telegram-бота (`botToken`, `chatID`). Усі ключові параметри можуть бути або жорстко задані в коді, або зчитані з конфігураційного файлу (наприклад, з SPIFFS), що забезпечує гнучкість у налаштуванні без потреби перепрошивки пристрою.

У функції `setup()` виконується ініціалізація послідовного порту, сенсорів DHT22 та VEML7700, підключення до Wi-Fi, а також налаштування HTTP-клієнта для комунікації з сервером. Вивід діагностичних повідомлень через Serial дозволяє здійснювати базове логування на етапі запуску. У разі потреби на цьому етапі також зчитуються додаткові конфігурації, наприклад, останнє збережене значення комфортної температури.

Головний цикл `loop()` реалізує періодичне зчитування показників із сенсорів у функції `readSensors()`, що повертає структуру з температурою, вологістю та рівнем освітленості. У разі наявності з'єднання з мережею (`WiFi.status() == WL_CONNECTED`) значення передаються на сервер у форматі JSON за допомогою HTTP POST-запиту. У відповідь система також може виконати HTTP GET-запит для отримання оновленої комфортної температури. Отримане значення, після перевірки на коректність, пересилається користувачу через Telegram-бота. В іншому випадку надсилається повідомлення про неможливість отримання рекомендації.

У разі втрати мережевого з'єднання система ініціює повторне підключення, а також передбачено збереження останніх зчитаних даних у локальну пам'ять. Між циклами опитування реалізовано фіксовану затримку (наприклад, 60 секунд).

Зчитування даних винесено в окрему функцію `readSensors()`, яка виконує перевірку валідності значень та повертає відповідну структуру. Надсилання повідомлень до Telegram організовано через функцію `sendToTelegram()`, що формує URL-запит до Telegram API та виконує його за умови наявності мережевого з'єднання. Усі помилки HTTP-запитів, збої з'єднання або зчитування сенсорів логуються, а у критичних випадках надсилається повідомлення користувачу.

Задля підвищення безпеки параметри з'єднання, ключі API та конфігураційні значення можуть бути збережені у окремому файлі `config.json` у файловій системі SPIFFS, або передані через компіляційні змінні (`#define`, `build_flags` у PlatformIO).

Прошивка реалізована у спосіб, що забезпечує легку розширюваність: додавання нових сенсорів можливе шляхом модифікації функції `readSensors()` та динамічного формування JSON. Крім того, передбачено можливість реалізації функцій автоматичного керування виконавчими пристроями (наприклад, реле), що може бути додано в майбутніх версіях прошивки без зміни базової структури коду.

```
JSON: {"temperature":27.90,"humidity":41.60,"light":0}
Відповідь сервера: 400
Отримано комфортну температуру: 21.4
Повідомлення надіслано в Telegram.
```

Рис. 2.5 – Наглядна робота ESP32

#### 2.4. Серверна частина на Flask

У цьому підрозділі описано структуру серверного коду, реалізацію маршрутів, налаштування періодичних завдань, обробку файлів, логіку конфігурації та обробку помилок.

### 2.4.1. Організація проєкту

Структура папок:

sensor.py – головний файл, який запускає Flask-сервер. Він відповідає за взаємодію з клієнтом, отримання даних та відображення результатів.

climate\_analyzer.py – модуль машинного навчання, який аналізує всі зібрані дані (температура, вологість, освітленість, прогноз погоди) та обчислює оптимальну температуру для підтримання комфортного клімату.

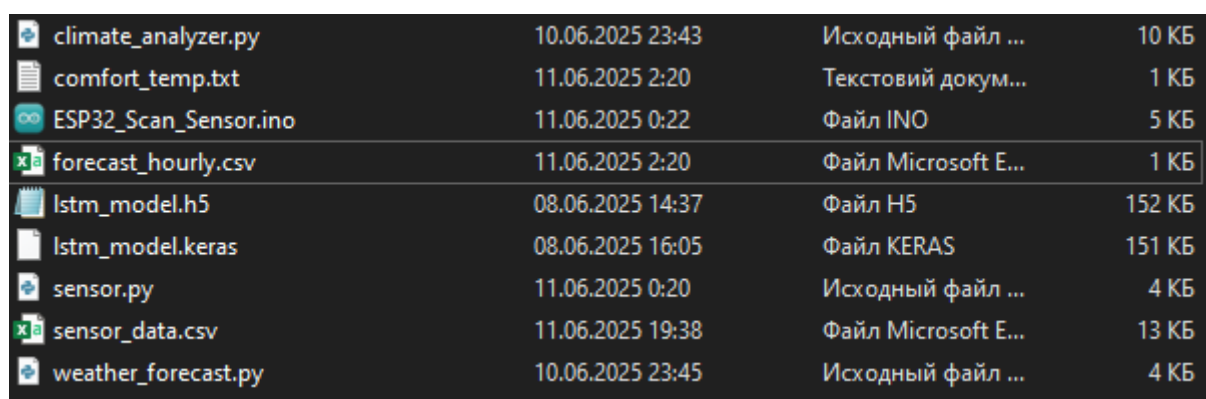
weather\_forecast.py – модуль прогнозу погоди. Отримує актуальні погодні дані з API або локальних файлів і надає інформацію про погодні умови на найближчі години.

sensor\_data.csv – файл, у якому накопичуються дані, зібрані з датчиків VEMML7700 (освітленість) і DHT22 (температура і вологість).

forecast\_hourly.csv – файл, який містить погодинний прогноз погоди. Застосовується для передбачення майбутніх умов та аналізу кліматичних тенденцій.

comfort\_temp.txt – текстовий файл, що містить останню обчислену рекомендовану температуру для створення комфортного середовища в приміщенні.

ESP32\_Scan\_Sensor.ino – скетч для мікроконтролера ESP32, який зчитує дані з фізичних датчиків і надсилає повідомлення (наприклад, попередження або рекомендації) у Telegram-бот.



Ім'я файлу	Дата	Тип файлу	Розмір
climate_analyzer.py	10.06.2025 23:43	Исходный файл ...	10 КБ
comfort_temp.txt	11.06.2025 2:20	Текстовий докум...	1 КБ
ESP32_Scan_Sensor.ino	11.06.2025 0:22	Файл INO	5 КБ
forecast_hourly.csv	11.06.2025 2:20	Файл Microsoft E...	1 КБ
lstm_model.h5	08.06.2025 14:37	Файл H5	152 КБ
lstm_model.keras	08.06.2025 16:05	Файл KERAS	151 КБ
sensor.py	11.06.2025 0:20	Исходный файл ...	4 КБ
sensor_data.csv	11.06.2025 19:38	Файл Microsoft E...	13 КБ
weather_forecast.py	10.06.2025 23:45	Исходный файл ...	4 КБ

Рис. 2.6 – Організація файлів проєкту

## 2.4.2. Ініціалізація Flask і APScheduler

Імпортуються необхідні бібліотеки та модулі:

- Flask — для створення веб-сервера.
- BackgroundScheduler і IntervalTrigger з пакету apscheduler — для запуску фонових задач з заданим інтервалом.
- atexit — для коректного завершення задач при зупинці сервера.
- datetime і os — для роботи з часом і системними ресурсами.
- Користувацькі модулі climate\_analyzer та weather\_forecast.

Потім ініціалізується Flask-додаток `app = Flask(__name__)` і визначається функція `start_periodic_tasks()`, яка створює планувальник BackgroundScheduler з урахуванням часової зони, після цього додає два періодичні завдання:

- `save_next_hour_forecast` — зберігає прогноз погоди на наступну годину.
- `run_analysis` — виконує аналіз кліматичних умов за зібраними даними.

Планувальник автоматично вимикається під час завершення роботи програми через `atexit`. При успішному запуску виводиться повідомлення: Scheduler запущено

У головному блоці програми: `if __name__ == '__main__':` спочатку запускається `start_periodic_tasks()` для активації фонових задач, потім запускається Flask-сервер з параметрами `host` і `port`, визначеними у конфігураційному файлі:

```
C:\My\KPI\Diploma\ESP32_Scan_Sensor>python sensor.py
2025-06-11 19:27:57.282430: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-06-11 19:28:05.583987: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
Scheduler запущено: tasks every 30 minutes
* Serving Flask app 'sensor'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.108:5000
Press CTRL+C to quit
```

Рис. 2.7 – Запуск Flask сервера

## 2.4.3. Маршрути Flask

Ендпоінт `POST /data` призначений для приймання даних із сенсорів освітленості VEMML7700 та температури/вологості DHT22, які надсилає мікроконтролер ESP32. Далі очікується JSON з полями `temperature`, `humidity`,

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		25

light, після чого дані парсяться та записуються у CSV-файл sensor\_data.csv у форматі дата\_та\_час, температура, вологість, освітленість

Також реалізований захист від помилок. Якщо JSON має некоректні або неповні дані — повертається статус 400 та відповідне повідомлення. У випадку помилки під час запису — повертається статус 500, при цьому деталі помилки логуються в консоль. А у разі успішного запису сервер повертає 200 ОК.

Ендпоінт GET /comfort\_temp дозволяє отримати останнє рекомендоване значення температури комфорту, яке попередньо зберігається у текстовому файлі comfort\_temp.txt. Значення зчитується з файлу. Після зчитування перевіряється, чи може бути коректно перетворене на float.

Якщо коректно отримати дані не вийшло, то починається обробка помилок. В ситуації коли файл відсутній або містить некоректне значення — повертається статус 500, і проблема логуються в консоль: не вдалося прочитати comfort\_temp.txt: ...

У випадку помилки ESP32 отримує відповідь про помилку, що дозволяє йому зреагувати через Telegram-бота повідомити про проблему, замість беззвучного ігнорування. Це є надійною практикою, адже допомагає виявляти та усувати реальні проблеми в системі, а не приховувати їх дефолтними значенням.

```
Записано: 2025-06-11 20:52:04, 24.0°C, 49.9%, 653.0 лк
192.168.0.101 - - [11/Jun/2025 20:52:04] "POST /data HTTP/1.1" 200 -
192.168.0.101 - - [11/Jun/2025 20:52:04] "GET /comfort_temp HTTP/1.1" 200 -
```

Рис. 2.8 – Коректна робота GET і POST

#### 2.4.4. Модулі прогнозу погоди і ML-аналіз

У файлі app.py логіка аналізу даних і формування прогнозу не дублюється, а делегується відповідним модулям. Імпортуються лише потрібні функції — run\_analysis з climate\_analyzer.py і save\_next\_hour\_forecast з weather\_forecast.py. Це дозволяє зберігати основний файл сервера компактним і зосередженим лише на налаштуванні маршрутизації та розкладу фонових задач.

climate\_analyzer.py містить реалізацію функції run\_analysis, яка має параметри send\_to\_esp і plot\_results. У контексті Flask-додатку параметр plot\_results не використовується, а send\_to\_esp встановлюється в False, щоб

функція не надсилала запитів до ESP32. Це зроблено для того, щоб сервер лише оновлював файл `comfort_temp.txt`, а ESP32 самостійно зчитував це значення через GET-запит до відповідного ендпоінта. Таким чином, роль надсилання даних перекладена на сам пристрій, що робить систему гнучкішою й менш залежною від прямої взаємодії.

`weather_forecast.py` реалізує функцію `save_next_hour_forecast`, яка щогодини оновлює файл `forecast_hourly.csv`, додаючи свіжі дані про прогноз погоди. Вона інтегрована у фоновий планувальник `APScheduler`, що забезпечує автоматичне й безперервне оновлення без втручання користувача.

#### 2.4.5. Обробка винятків та логування

У всіх функціях, що працюють з файловою системою або здійснюють мережеві виклики, реалізовано обробку винятків через конструкції `try/except`. Це дозволяє уникнути неочікуваного завершення роботи у разі помилки та забезпечує контроль над ситуацією. Для відлагодження та зручного аналізу помилок усі повідомлення логуються через `print()`, включаючи позначку часу та опис самої помилки. Такий підхід забезпечує базовий рівень прозорості на етапі розробки.

Хоча на даному етапі прототипу вивід у консоль є достатнім, у подальшому можливо підключити повноцінний модуль `logging` із підтримкою ротації логів через `RotatingFileHandler`. Це дозволить зберігати історію подій, контролювати розмір лог-файлів і організувати журналювання на рівні продакшену.

#### 2.4.6. Перевірка та ініціалізація файлів

Під час старту Flask-сервера виконується попередня перевірка наявності ключових файлів: `sensor_data.csv`, `forecast_hourly.csv` та `comfort_temp.txt`. Якщо якийсь із них відсутній, він автоматично створюється як порожній, що гарантує стабільність подальшої роботи системи та уникнення помилок доступу до неіснуючих ресурсів. Це реалізовано за допомогою простого проходження по списку шляхів до файлів. Для кожного з них перевіряється наявність, і, якщо потрібно, виконується ініціалізація з повідомленням у консоль.

Хоча в CSV-файлах можна додати заголовки з назвами стовпців (наприклад: `timestamp,temperature,humidity,light`), у випадку простого накопичення часових рядів це не є обов'язковим. Система орієнтована на зручність парсингу, а не на ручне опрацювання, тому достатньо послідовного запису рядків із роздільниками.

#### 2.4.7. Налаштування APScheduler та часові зони

Для забезпечення точності періодичних завдань у проєкті використовується APScheduler з підтримкою часової зони, що задається через `config.TIMEZONE` (наприклад, `Europe/Kiev`). Це дозволяє синхронізувати `start_date` та вести коректне логування з урахуванням локального часу. Для реальної експлуатації рекомендовано встановити інтервал у 30 хвилин, але під час розробки або тестування його легко змінити, наприклад, на 5 хвилин — це прискорює перевірку функціоналу.

```
from apscheduler.schedulers.background import BackgroundScheduler
from apscheduler.triggers.interval import IntervalTrigger
```

Рис. 2.9 Підключення бібліотек APScheduler

Запуск планувальника виконується ще до старту Flask-сервера, що дозволяє йому працювати у фоновому режимі паралельно з основним HTTP-сервісом. Завдяки цьому всі періодичні задачі, такі як збереження прогнозу погоди чи аналіз кліматичних даних, відбуваються автоматично та без впливу на обробку запитів REST API.

Змін.	Арк.	№ докум.	Підпис	Дата

```

def start_periodic_tasks():
    scheduler = BackgroundScheduler(timezone="Europe/Kiev")
    # Викликати save_next_hour_forecast кожні 30 хвилин, перший запуск одразу
    scheduler.add_job(
        save_next_hour_forecast,
        trigger=IntervalTrigger(minutes=30, start_date=datetime.now()),
        id="forecast_job"
    )
    # Викликати run_analysis кожні 30 хвилин, перший запуск одразу
    scheduler.add_job(
        run_analysis,
        trigger=IntervalTrigger(minutes=30, start_date=datetime.now()),
        id="analysis_job"
    )
    scheduler.start()
    # При завершенні програми зупинити scheduler
    atexit.register(lambda: scheduler.shutdown(wait=False))
    print("✅ Scheduler запущено: tasks every 30 minutes")

```

Рис. 2.10 Застосування APScheduler в роботі сервера

#### 2.4.8. Конфігурація Telegram-бота на сервері

З метою забезпечення зворотного зв'язку із користувачем у системі передбачено можливість надсилання повідомлень із серверної частини за допомогою Telegram-бота. Такий механізм дає змогу оперативно інформувати користувача про виникнення аномальних ситуацій, помилки при виконанні модулів прогнозування або інші важливі події без необхідності постійного моніторингу інтерфейсу.

Конфігураційні параметри, необхідні для авторизації бота, включають токен доступу до Telegram API та ідентифікатор цільового чату. Вони зберігаються у конфігураційному модулі config.py, що дозволяє централізовано керувати налаштуваннями без внесення змін до основного програмного коду. Такий підхід спрощує супровід проекту, забезпечує кращу безпеку (наприклад, шляхом винесення чутливих даних за межі репозиторію) та уможливорює розгортання системи в різних середовищах.

Функція надсилання повідомлень реалізована як окрема утиліта, що використовує HTTP-запит до Telegram API. Вона може бути інтегрована безпосередньо у головний файл серверного застосунку або винесена в окремий модуль для повторного використання.

З міркувань надійності передбачена обробка виключень, логування помилок та обмеження часу очікування відповіді. Практичне використання Telegram-бота полягає у виклику функції надсилання повідомлень із серверних процедур.

Зокрема, у модулі аналізу кліматичних даних (`run_analysis`) бот може повідомляти про критичні ситуації — наприклад, якщо протягом тривалого часу не оновлювався прогноз погоди або якість моделі не відповідає заданим метрикам. З метою уникнення надмірної кількості сповіщень впроваджено обмеження частоти надсилання повідомлень або додаткову логіку фільтрації подій.

Таким чином, інтеграція Telegram-бота у серверну частину дозволяє реалізувати базову систему сповіщень без потреби у складному інтерфейсі користувача. Вона забезпечує інформативність, зручність та розширює можливості моніторингу стану системи у реальному часі.

#### 2.4.9. Обробка багатопоточності та конкуренції доступу до файлів

У процесі реалізації серверної частини було враховано, що модуль планування завдань `APScheduler` виконує функції у фонових потоках, що потенційно може призвести до одночасного доступу до спільних ресурсів — зокрема, до файлових структур з даними. Така ситуація виникає, наприклад, коли один з модулів записує нові показники сенсорів у файл `sensor_data.csv`, у той час як аналітичний модуль паралельно читає цей файл для підготовки вхідних даних до моделі прогнозування.

Зважаючи на те, що операції запису виконуються у режимі додавання рядків (`append`), а читання — у вигляді повного зчитування файлу, ймовірність критичних конфліктів при роботі системи в стандартному режимі є низькою. Тим не менш, для забезпечення надійності та запобігання потенційним помилкам зчитування або пошкодження даних передбачено кілька стратегій синхронізації доступу.

Зокрема, у простих випадках доцільно використовувати механізми синхронізації на рівні потоків, наприклад, блокування з використанням

threading.Lock у Python. Впровадження такого підходу дозволяє гарантувати, що під час читання або запису даних жоден інший потік не отримає доступ до відповідного файлу. В якості альтернативи можна застосовувати атомарні операції на рівні файлової системи — наприклад, здійснювати запис у тимчасовий файл із подальшим перейменуванням, що гарантує цілісність навіть у разі несподіваного завершення операції.

Хоча в рамках даного проєкту система працює з невеликими обсягами даних і виконує обробку з відносно низькою частотою, у звіті спеціально підкреслено важливість врахування паралельного доступу при подальшому масштабуванні системи або перенесенні її на багатопроцесорні середовища. Задля узагальнення рекомендовано винести критичні операції з файлами у спеціалізовані функції-обгортки, що забезпечують узгодженість доступу і спрощують подальшу підтримку системи.

## 2.5. Модуль прогнозу погоди

У рамках архітектури інтелектуальної кліматичної системи окремий функціональний блок реалізує отримання прогнозу погоди із зовнішнього джерела. Цей компонент винесено в окремий модуль `weather_forecast.py`, що забезпечує незалежність логіки прогнозування від решти системи та спрощує її модифікацію.

### 2.5.1. Вибір джерела метеоданих

Для отримання метеорологічної інформації було обрано безкоштовне публічне API, зокрема сервіс Open-Meteo, що не вимагає автентифікаційного ключа з платіжною прив'язкою та не накладає жорстких обмежень на частоту запитів. Такий вибір є доцільним для прототипу навчального або дослідницького проєкту, оскільки дозволяє уникнути прив'язки до комерційних сервісів на ранньому етапі розробки.

```

def get_next_hour_forecast():
    """
    повертає (time_str, temperature) для наступної повної години,
    або (None, None) якщо не знайдено.
    """
    url = (
        f"https://api.open-meteo.com/v1/forecast?"
        f"latitude={LAT}&longitude={LON}&hourly=temperature_2m&timezone=Europe/Kiev"
    )
    try:
        response = requests.get(url, timeout=10)
        response.raise_for_status()
        data = response.json()
    except Exception as e:
        print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}] ❌ Помилка HTTP при отриманні прогнозу: {e}")
        return None, None

    times = data.get('hourly', {}).get('time', [])
    temps = data.get('hourly', {}).get('temperature_2m', [])
    if not times or not temps or len(times) != len(temps):
        print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}] ❌ Невірна структура відповіді прогнозу")
        return None, None

    next_hour = (datetime.now() + timedelta(hours=1)).replace(minute=0, second=0, microsecond=0)
    next_hour_str = next_hour.strftime('%Y-%m-%dT%H:%M')
    for t, temp in zip(times, temps):
        # t завжди у форматі 'YYYY-MM-DDThh:mm'
        if t.startswith(next_hour_str):
            return t, temp
    return None, None

```

Рис. 2.11 реалізація зчитування даних з Open-Meteo

Конфігурація API, зокрема базовий URL, координати географічного розташування (широта, довгота) та бажані погодні параметри, зберігається у відповідних файлах проєкту. Це забезпечує гнучкість: у разі зміни джерела даних або перенесення системи до іншого регіону, достатньо змінити параметри конфігурації без втручання в логіку обробки запитів.

### 2.5.2. Функція отримання прогнозу на наступну годину

Для забезпечення коректної роботи алгоритму аналізу мікроклімату у проєкті реалізовано функцію `get_next_hour_forecast()`, яка здійснює запит прогнозних метеоданих на одну годину вперед.

Ціль функції: отримати числове значення температури повітря на найближчу цілу годину, що є ключовим параметром для розрахунку комфортного температурного діапазону в приміщенні.

Основні етапи роботи функції:

1. Формування запиту до API.

В HTTP-запиті вказуються географічні координати місця (широта та довгота), часовий пояс Europe/Kiev, а також параметри для отримання годинного прогнозу температури (temperature\_2m).

## 2. Обробка відповіді.

У відповіді API очікується структура JSON, що містить два основних масиви: time (часові мітки прогнозу у форматі ISO 8601) та temperature\_2m (відповідні значення температури повітря).

## 3. Визначення часу прогнозу.

Функція обчислює поточний системний час, додає одну годину та округлює результат до початку години (тобто до формату без хвилин та секунд). Далі формується рядок часу у форматі YYYY-MM-DDThh:00, що відповідає формату часу в масиві time.

## 4. Пошук значення температури.

Серед елементів масиву time здійснюється пошук прогнозу на потрібну годину. Якщо відповідний індекс знайдено, повертається кортеж (forecast\_time, forecast\_temp). Якщо прогноз відсутній або час не знайдено — функція повертає (None, None).

### 2.5.3. Логування прогнозу

У рамках реалізації модуля прогнозування передбачено збереження отриманих прогнозних значень у вигляді окремого історичного журналу. Це дозволяє не лише використовувати прогнози для подальшого аналізу, а й забезпечує прозорість роботи системи та можливість її відлагодження.

Основну роль у цьому процесі відіграє функція save\_next\_hour\_forecast(), яка викликається періодично згідно з графіком, заданим у планувальнику. Після звернення до зовнішнього погодного API функція отримує час виконання запити (current\_time), прогнозований момент у майбутньому (forecast\_time) та значення температури на відповідну годину. У випадку, якщо обидва значення є коректними, формується рядок у CSV-форматі, що містить дату й час запити, цільовий прогнозний момент та відповідну температуру. Такий рядок додається до файлу forecast\_hourly.csv.

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		33

```

def save_next_hour_forecast():
    """
    Отримує прогноз для наступної години і записує у CSV_FILENAME:
    рядок [current_time, forecast_time, temperature].
    """
    forecast_time, temperature = get_next_hour_forecast()
    current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    if forecast_time and temperature is not None:
        try:
            # Якщо файл може не існувати, режим 'a' створить його
            with open(CSV_FILENAME, mode='a', newline='', encoding='utf-8') as file:
                writer = csv.writer(file)
                writer.writerow([current_time, forecast_time, temperature])
            print(f"[{current_time}] ✓ Записано прогноз: {forecast_time} | {temperature}°C")
        except Exception as e:
            print(f"[{current_time}] ✗ Помилка при записі у файл {CSV_FILENAME}: {e}")
        else:
            print(f"[{current_time}] ✗ Не вдалося отримати прогноз на наступну годину.")

```

Рис. 2.12 Реалізації функції save\_next\_hour\_forecast()

Запис здійснюється у режимі додавання (append), що дозволяє автоматично створювати файл, якщо він не існував раніше. У консолі виводиться службове повідомлення з інформацією про доданий прогноз, включаючи часову мітку та значення температури. У разі, якщо прогноз не було отримано (наприклад, через помилку мережевого з'єднання або некоректну відповідь API), система логуватиме відповідне повідомлення із зазначенням дати й часу невдалої спроби.

Збереження історії прогнозів є важливим як для відстеження стабільності роботи модуля, так і для додаткового аналізу — зокрема, порівняння прогнозованих і фактичних значень температури, оцінки точності моделі або вдосконалення алгоритмів прогнозування у майбутньому. Така практика підвищує прозорість функціонування системи і закладає основу для побудови надійних моделей контролю якості.

#### 2.5.4. Налаштування періодичності

Оновлення прогнозу зовнішньої температури реалізовано за допомогою періодичного запуску відповідного модуля через планувальник завдань APScheduler. Функція save\_next\_hour\_forecast() викликається із заданим інтервалом, наприклад, кожні 30 хвилин. Такий підхід дозволяє не лише підтримувати актуальність прогнозової інформації, а й накопичувати історію змін

прогнозу, що особливо корисно для подальшого аналізу або оцінки точності джерела даних.

Періодичний запуск із частотою, вищою за одну годину, дає змогу зафіксувати динаміку оновлень прогнозів: один і той самий прогнозований часовий відрізок може кілька разів оновлюватися зовнішнім API, а система, відповідно, зберігатиме ці зміни у файлі `forecast_hourly.csv`. Це дозволяє простежити, наскільки стабільним є прогноз або як змінюються дані при наближенні до прогнозованої події.

У модулі аналізу кліматичних умов (`run_analysis`) під час розрахунку комфортної температури використовується останній доступний запис із прогнозного файлу, що відповідає найближчій майбутній годині. Це забезпечує адаптивність алгоритму: навіть якщо поточний прогноз ще не оновився або вже частково застарів, система все одно оперує найсвіжішими доступними даними. Така стратегія підвищує стійкість до можливих затримок або збоїв на стороні погодного сервісу та забезпечує максимально релевантну інформацію для прийняття рішень.

#### 2.5.5. Обробка обмежень і відмов

У випадках, коли зовнішній API прогнозу недоступний або повертає некоректні дані, система зберігає працездатність. Алгоритм розрахунку комфортної температури не залежить критично від наявності прогнозу, а адаптується до поточних умов, використовуючи лише внутрішні показники з сенсорів. Якщо актуальний прогноз відсутній, аналіз може бути проведений на основі останнього збереженого значення або взагалі без врахування прогнозованої температури, що забезпечує стійкість до зовнішніх збоїв.

Помилки, пов'язані з отриманням прогнозу, фіксуються у журналі подій. Таке логування дає змогу виявляти частоту відмов або нестабільну поведінку стороннього API. На основі цієї інформації можливе прийняття рішень щодо зміни джерела прогнозів, корекції інтервалів оновлення або запровадження додаткових стратегій відновлення, таких як повторна спроба отримання даних через певний час.

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		35

## 2.6. ML-модуль та алгоритм розрахунку комфортної температури

У цьому підрозділі представлено реалізацію програмного модуля `climate_analyzer.py`, що виконує ключові завдання аналітичної частини системи. Зокрема, в межах цього файлу організовано повний цикл обробки кліматичних даних: підготовку вхідної інформації, тренування та використання моделі машинного навчання, формування прогнозу внутрішньої температури, а також розрахунок комфортної температури з урахуванням різних факторів.

Реалізація охоплює як первинну обробку даних, зчитаних із сенсорів та зовнішніх джерел, так і підготовку вибірки для моделі — включно з масштабуванням, формуванням часових ознак та обробкою пропущених значень. Тренування моделі здійснюється локально з можливістю збереження її параметрів для повторного використання без потреби у перенавчанні при кожному запуску. Крім того, модуль виконує обчислення цільової температури комфорту, яка може використовуватися як рекомендація для користувача або автоматичних регуляторів клімату.

Для оцінки точності прогнозової моделі температури було використано метрику середньоквадратичної помилки (MSE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

де  $y_i$  — фактичне значення температури,  $\hat{y}_i$  — прогнозоване,  $n$  — кількість вимірів

### 2.6.1. Збір і підготовка даних

Підготовка даних є ключовим етапом перед використанням моделей прогнозування. У модулі `climate_analyzer.py` реалізовано механізм обробки як внутрішніх показників, отриманих із сенсорів, так і зовнішніх прогнозних значень температури, що забезпечує комплексний підхід до формування вхідної вибірки.

Внутрішні дані завантажуються з локального CSV-файлу `sensor_data.csv`, що містить часові ряди значень температури, вологості та освітленості. Для обробки використовується бібліотека `pandas`, яка дозволяє зручно структурувати

таблицю та проводити фільтрацію. Поле дати та часу перетворюється до стандартного типу `datetime`, що дає змогу ефективно оперувати часовими ознаками. Додатково проводиться фільтрація некоректних або аномальних значень: температура повинна знаходитися у фізично допустимому діапазоні (0–50 °C), вологість — у межах 0–100 %, а освітленість не повинна бути від’ємною. Таким чином формується чистий набір даних, придатний для моделювання.

Для поліпшення якості моделі додаються похідні ознаки, які можуть містити приховані кореляції — зокрема, година доби, що враховує добовий ритм зміни температури, а також місяць або сезон, які можуть бути корисними при довготривалому спостереженні. У випадках, коли обсяг доступних даних надто малий (наприклад, менший за довжину послідовності, необхідну для рекурентної нейромережі), система автоматично визначає неможливість побудови прогнозу та переходить у режим очікування накопичення нових даних.

Паралельно з обробкою внутрішньої інформації здійснюється зчитування зовнішніх даних прогнозу з файлу `forecast_hourly.csv`. Цей файл містить хронологічні прогнози температури, отримані з погодного API. Враховуючи час запуску аналізу, система відбирає найбільш актуальний прогноз на найближчу доступну годину — або шляхом пошуку першого запису, що відповідає майбутньому, або шляхом вибору останнього з тих, що все ще актуальні. Якщо прогноз недоступний або застарів, цей факт фіксується, і модуль переходить до обчислень без залучення зовнішнього джерела.

Завдяки поєднанню внутрішніх сенсорних даних і зовнішніх прогнозів забезпечується більш гнучка та адаптивна підготовка вибірки, що дозволяє підвищити точність моделей та забезпечити стабільну роботу в умовах обмеженого або неповного потоку інформації.

### 2.6.2. Формування даних для LSTM

Для прогнозування внутрішньої температури в системі використовується модель типу LSTM (Long Short-Term Memory), що належить до класу рекурентних нейронних мереж і добре пристосована до обробки часових рядів.

Змін.	Арк.	№ докум.	Підпис	Дата

Перед тренуванням такої моделі необхідно сформувати навчальну вибірку у вигляді послідовностей фіксованої довжини.

На підставі попередньо очищених даних обираються ключові ознаки — температура, вологість та рівень освітленості. Ці параметри нормалізуються до інтервалу  $[0, 1]$  за допомогою методу `MinMaxScaler`. Масштабування даних є обов'язковим кроком для коректного функціонування нейронної мережі, оскільки воно пришвидшує збіжність під час навчання та зменшує ризик домінування однієї ознаки над іншими.

Подальше формування вибірки виконується за допомогою ковзаючого вікна фіксованої довжини. Задається параметр `SEQ_LEN`, який визначає кількість попередніх часових точок, що використовуються як вхід для прогнозу наступного значення температури. На кожному кроці формуються вектори ознак `X_seq` та відповідні значення-цілі `y_seq`, що становлять температуру у наступний момент часу після поточної послідовності. Зібрані послідовності конвертуються до масивів `NumPy`, які можуть бути подані на вхід моделі `LSTM`.

Якщо обсяг вхідних даних недостатній для формування хоча б однієї повної послідовності, модель не тренується. У такому випадку система переходить до спрощеної стратегії, яка використовує останнє вимірне значення як прогноз — це дозволяє зберегти базову функціональність системи навіть у разі браку даних.

Для контролю якості моделі дані розділяються на навчальну та тестову вибірки. Зазвичай 80% масиву використовується для тренування, а решта — для валідації. У випадках, коли доступна лише невелика кількість прикладів, оцінювання точності прогнозу проводиться з обережністю або може бути пропущене, якщо перевага надається стабільності роботи над формальними метриками якості.

### 2.6.3. Завантаження або тренування моделі

Після формування вибірки система визначає, чи можливо використати раніше збережену модель, або ж потрібно здійснити нове тренування. Для цього передбачено перевірку наявності моделі у файлі `lstm_model.keras`. У разі

успішного завантаження моделі за допомогою функції `load_model`, подальше тренування пропускається, що суттєво зменшує час запуску.

Якщо модель відсутня або не підлягає завантаженню, створюється нова нейронна мережа, що має стандартну архітектуру з одним шаром LSTM і вихідним щільним шаром. Використовується оптимізатор `adam` та функція втрат `mean squared error`:

```
lstm_model = Sequential([
    Input(shape=(SEQ_LEN, 3)),
    LSTM(50, activation='relu'),
    Dense(1)
])
lstm_model.compile(optimizer='adam', loss='mse')
lstm_model.fit(X_train_lstm, y_train_lstm, epochs=20, verbose=1)
lstm_model.save(MODEL_PATH)
```

Рис. 2.13 – Створення нової нейронної мережі

Кількість нейронів (50) і кількість епох (20) вибрано на основі практичного компромісу між якістю моделі та обмеженнями швидкодії системи. Завдяки цьому забезпечується базовий рівень точності, при якому модель не перенавчається і може працювати стабільно на обмежених апаратних ресурсах.

Після тренування або завантаження моделі здійснюється оцінка її якості на відкладеній вибірці. Для цього використовується функція `evaluate`, яка обчислює середньоквадратичну помилку (MSE). Для інтерпретації результатів помилка переводиться у форму кореневої середньоквадратичної (RMSE), що дозволяє оцінити точність прогнозу у градусах Цельсія. При надмірно високому значенні RMSE у логах фіксується попередження, і система може бути автоматично переведена у режим спрощеного прогнозування на основі останнього доступного значення.

У випадках, коли обсяг навчальної вибірки є недостатнім для надійного тренування або оцінки моделі, алгоритм приймає рішення про відмову від машинного навчання та переходить до евристичного варіанту прогнозу. Цей механізм дозволяє зберегти працездатність системи в умовах браку даних або на етапі первинного запуску.

#### 2.6.4. Прогноз внутрішньої температури

Прогнозування температури всередині приміщення здійснюється за допомогою раніше натренованої моделі типу LSTM, якщо така доступна. Основною метою цього етапу є оцінка температурного значення на найближчий момент часу, враховуючи поточну динаміку змін кліматичних параметрів.

У випадку наявності працездатної моделі прогноз формується на основі останніх вимірних даних, які були попередньо нормалізовані. Із цього масиву виділяються SEQ\_LEN останніх записів, які використовуються як вхідний приклад для моделі

#### 2.6.5. Алгоритм розрахунку комфортної температури

Комфортна температура розраховується з урахуванням впливу вологості та освітленості за евристичною формулою:

$$T_{\text{комфорт}} = T_{\text{внутр}} + \alpha(H_{\text{оптим}} - H_{\text{факт}}) + \beta(L_{\text{оптим}} - L_{\text{факт}})$$

значення  $\alpha$  та  $\beta$  обрані емпірично для демонстрації чутливості температури до зміни параметрів середовища.

Для підвищення адаптивності системи до реальних умов середовища реалізовано функцію `calculate_comfort_temperature()`, яка обчислює рекомендовану комфортну температуру з урахуванням кількох параметрів: поточної внутрішньої температури, вологості, рівня освітленості, а також прогнозованої зовнішньої температури. Такий підхід базується на поєднанні простих емпіричних правил, які враховують фізіологічне сприйняття мікроклімату, з актуальними прогнозними даними, що дає змогу персоналізувати рекомендації.

Базове значення комфортної температури визначено на рівні 22 °C, що відповідає середньому значенню, рекомендованому для житлових і офісних приміщень. Далі значення коригується відповідно до поточних умов.

Вологість повітря істотно впливає на теплові відчуття людини. При підвищеній вологості (> 70 %) виникає ефект "задушливості", тому комфортна температура знижується на певну величину (наприклад, 0.5–1 °C). У разі надто

низької вологості (< 30 %) часто виникає відчуття сухості та прохолоди — в такому випадку базове значення температури підвищується для компенсації.

Рівень освітлення також має вплив: при низькій освітленості (< 300 люкс), особливо в похмурі дні, температура сприймається як нижча — тому рекомендується невелике підвищення. Натомість при яскравому природному освітленні візуальний і тепловий комфорт часто асоціюється з вищою температурою, що виправдовує її коригування в бік зниження.

Прогноз зовнішньої температури дозволяє врахувати очікувані теплові потоки у майбутньому. Якщо температура на вулиці найближчим часом перевищить 27 °С, система передбачає додаткове прогрівання приміщення, тому комфортна температура відповідно знижується. У випадку очікуваного похолодання (< 18 °С) рекомендована температура трохи підвищується для збереження відчуття тепла.

Після всіх коригувань значення проходить перевірку на відповідність допустимому діапазону. Встановлено межі від 19 °С до 25 °С, що відповідає типовим нормам комфорту. У разі виходу за межі, значення обрізається до найближчої межі. Для зручності виводу та подальшого порівняння фінальний результат округлюється до 0.1 °С.

Таким чином, розрахунок комфортної температури реалізовано як адаптивну функцію, що враховує одночасно кілька контекстних факторів, з можливістю подальшої модифікації. Такий гібридний підхід поєднує правила, засновані на власних спостереженнях і базових принципах теплового комфорту, з динамічними даними прогнозу, забезпечуючи гнучкість і наближеність до реальних потреб користувача.

#### 2.6.6. Оновлення файлу комфортної температури

Після обчислення комфортної температури результат зберігається у файл `comfort_temp.txt`. Запис здійснюється у текстовому форматі у режимі перезапису (`write`), при цьому у файл виводиться лише саме числове значення, без додаткових символів або одиниць вимірювання. Такий підхід забезпечує сумісність із мікроконтролером ESP32, який періодично зчитує вміст цього

файлу для отримання актуальної рекомендації. Запис у файл супроводжується логуванням до консолі з фіксацією часу та значення, що було збережене, наприклад: [2025-06-11 10:00:00] Обчислено комфортну температуру: 22.5 °C.

#### 2.6.7. Логування і обробка виключень

З огляду на те, що модуль аналізу даних виконується автоматично з певною періодичністю та працює з різнорідними джерелами інформації, важливим аспектом його реалізації є забезпечення стійкості до помилок. Для цього критичні ділянки коду обгортаються в конструкції try/except, що дозволяє фіксувати помилки без порушення загальної працездатності системи.

Зокрема, при зчитуванні файлу сенсорних даних `sensor_data.csv` можуть виникати помилки, пов'язані з порушенням формату, відсутністю файлу або тимчасовими обмеженнями доступу. У таких випадках помилка фіксується в логах, після чого функція переривається з повідомленням, наприклад:

except Exception as e:

```
print(f"Помилка читання sensor_data.csv: {e}")  
return
```

Якщо не вдалося отримати прогноз зовнішньої температури, система продовжує роботу: у функцію обчислення комфортної температури передається `forecast_temp=None`, що активує механізм роботи без прогнозу — лише на основі внутрішніх показників. Це дозволяє зберегти основний функціонал у разі збоїв зовнішнього API.

Тренування моделі LSTM також може бути перервано через технічні обмеження, наприклад, нестачу оперативної пам'яті або помилки під час виклику `fit()`. Такі винятки також обробляються з переходом до резервної стратегії прогнозування, яка базується на останньому відомому значенні температури. При цьому у логах фіксується характер помилки та повідомлення про активацію fallback-режиму.

Оцінка точності моделі після навчання супроводжується обчисленням метрик MSE та RMSE. Якщо значення RMSE є надто високим (відносно

очікуваної точності), у журналі фіксується попередження про низьку якість моделі, після чого система також може перейти до спрощеного прогнозування.

Окремо обробляються винятки під час запису результату комфортної температури у файл `comfort_temp.txt`. Якщо файл не вдалося створити або перезаписати (наприклад, через відсутність прав доступу або збої файлової системи), відповідне повідомлення виводиться в лог. Така обережна обробка винятків дозволяє уникнути аварійного завершення процесу та забезпечує стабільну роботу модуля навіть у складних умовах.

#### 2.6.8. Розширюваність ML-модуля

Проектована архітектура модуля машинного навчання враховує можливості масштабування та подальшого розвитку, що дозволяє адаптувати систему до змін у даних або вимогах користувача.

Підтримка альтернативних моделей.

Кодова база організована таким чином, що реалізацію нової моделі — зокрема, алгоритму Random Forest або іншої класичної ML-моделі — можна виконати як окрему гілку (модуль або клас), не змінюючи логіку існуючої нейронної мережі LSTM. Це дозволяє проводити незалежне навчання, тестування і порівняння результатів різних моделей за стандартними метриками (наприклад, MSE, MAE). У межах дипломного звіту зазначається, що в умовах обмеженого обсягу історичних даних класичні алгоритми іноді демонструють вищу стабільність і точність, ніж складніші моделі на основі глибокого навчання. Можливість розширення ознак.

Передбачена гнучка структура обробки вхідних даних дозволяє без суттєвих змін розширювати набір ознак у навчальному DataFrame. Зокрема, у перспективі можливо включення додаткових змінних: історія зовнішньої температури, час доби, день тижня, присутність мешканців у приміщенні, дані з CO<sub>2</sub>-датчиків тощо. Додавання таких факторів покращить контекстуальне розуміння моделі й потенційно підвищить точність прогнозу.

Підтримка відкладеного навчання.

У випадках, коли навчання моделі вимагає значного часу або ресурсів, передбачено реалізацію механізму періодичного тренування — раз на добу або тиждень. У проміжках між перезапусками використовується попередньо збережена модель, що забезпечує постійну доступність прогнозного функціоналу без навантаження на систему. У звіті також описано можливість гнучкого налаштування частоти оновлень з урахуванням обчислювальних обмежень.

## 2.7. Інтеграція компонентів і періодичний запуск

У цьому розділі представлено принципи організації взаємодії між функціональними модулями системи, реалізацію механізмів періодичного виконання завдань, а також заходи, спрямовані на забезпечення надійності та безперервної роботи програмного комплексу. Особливу увагу приділено архітектурним рішенням, що забезпечують модульність, зручність масштабування і стійкість системи до збоїв зовнішніх сервісів.

### 2.7.1. Налаштування APScheduler у Flask

Для забезпечення періодичного виконання ключових функцій системи, у серверну частину інтегровано планувальник APScheduler, який ініціалізується у режимі BackgroundScheduler. Часовий пояс встановлено як "Europe/Kiev", що дозволяє коректно синхронізувати часові мітки при логуванні, збереженні прогнозу та виконанні розрахунків.

У рамках планувальника реєструються дві задачі з унікальними ідентифікаторами з метою уникнення їх дублювання:

- `forecast_job` — періодично викликає функцію `save_next_hour_forecast()`, яка відповідає за отримання та збереження прогнозу погоди;
- `analysis_job` — періодично викликає функцію `run_analysis()`, що здійснює розрахунок рекомендованої температури на основі зібраних даних.

Змін.	Арк.	№ докум.	Підпис	Дата

```

# Викликати save_next_hour_forecast кожні 30 хвилин, перший запуск одразу
scheduler.add_job(
    save_next_hour_forecast,
    trigger=IntervalTrigger(minutes=30, start_date=datetime.now()),
    id="forecast_job"
)
# Викликати run_analysis кожні 30 хвилин, перший запуск одразу
scheduler.add_job(
    run_analysis,
    trigger=IntervalTrigger(minutes=30, start_date=datetime.now()),
    id="analysis_job"
)

```

Рис 2.14 Реалізація викликів функцій обробки даних за заданий період

Інтервал виконання кожної задачі задається через змінну-константу, наприклад, `APScheduler_INTERVAL_MINUTES = 30`. Для обох задач параметр `start_date` встановлено як `datetime.now()`, що забезпечує їх запуск одразу після старту серверу, навіть якщо історія даних відсутня. Це дозволяє одразу створити необхідні службові файли (зокрема, `comfort_temp.txt`) та ініціалізувати базовий стан системи.

Для коректного завершення планувальника під час зупинки роботи сервера зареєстровано обробник завершення через `atexit.register(scheduler.shutdown)`, що гарантує звільнення ресурсів та запобігає фоновому зависанню процесу `APScheduler`.

### 2.7.2. Послідовність роботи

Після запуску серверної частини системи на Flask ініціюється автоматичне виконання ключових функцій завдяки інтегрованому планувальнику `APScheduler`. Усі компоненти працюють у координованому циклі, що забезпечує безперервне оновлення даних та підтримку актуальних рекомендацій.

#### 1. Початковий запуск:

- Після старту Flask-сервера планувальник негайно викликає функцію `save_next_hour_forecast()`.
- Якщо прогноз погоди успішно отримано з API, він логуються у файл `forecast_hourly.csv`.

#### 2. Початковий аналіз:

- Відразу після отримання прогнозу викликається функція `run_analysis()`, яка виконує такі дії:

- Зчитує дані з `sensor_data.csv` (який може бути порожнім при першому запуску) та `forecast_hourly.csv`.

- Якщо обсяг наявних даних недостатній для повноцінного аналізу, активується fallback-механізм — розрахунок комфортної температури проводиться без участі ML-моделі.

- Результуюче значення комфортної температури записується у файл `comfort_temp.txt`.

### 3. Взаємодія з мікроконтролером ESP32:

- Під час першого циклу після запуску прошивки ESP32 надсилає POST-запит із сенсорними даними на сервер.

- Потім здійснює GET-запит на маршрут `/comfort_temp`, отримує обчислену комфортну температуру та надсилає повідомлення у Telegram.

### 4. Регулярне оновлення:

- Надалі з інтервалом у 30 хвилин APScheduler повторно виконує функції `save_next_hour_forecast()` і `run_analysis()`, забезпечуючи актуальність прогнозу та рекомендацій.

- ESP32, працюючи з іншим (коротшим) інтервалом, наприклад 1 хвилина, періодично надсилає нові сенсорні дані та отримує поточне значення рекомендованої температури.

### 5. Механізми стійкості (Fallback-логіка):

- У разі недоступності зовнішнього API погоди, модуль `run_analysis()` використовує евристичні правила для розрахунку комфортної температури без участі прогнозу.

- Якщо файл `sensor_data.csv` містить недостатню кількість записів для роботи LSTM-моделі, прогноз внутрішньої температури базується на останньому відомому значенні або емпіричному шаблоні.

- Таким чином, система гарантує наявність коректного значення у `comfort_temp.txt` навіть у разі збоїв чи браку вхідних даних.

### 2.7.3. Обробка виключень у періодичних завданнях

Щоб гарантувати безперервну роботу планувальника APScheduler, у функціях `save_next_hour_forecast` та `run_analysis` реалізовано конструкції `try/except`, які обробляють усі можливі винятки. У разі виникнення помилки вона логуються у консоль або лог-файл, після чого функція завершує виконання без генерації виключення на рівень планувальника.

Аналогічна обгортка використовується у `save_next_hour_forecast`, де особливо важливо обробити некоректну відповідь API або проблеми з мережею. Такий підхід забезпечує стійкість системи до одиничних збоїв та запобігає повному припиненню виконання задач.

### 2.7.4. Налаштування інтервалів та моніторинг

Інтервал виконання завдань `forecast_job` та `analysis_job` задається у конфігурації системи та може бути легко змінений відповідно до обраного сценарію використання. Інтервал 15 хвилин — для чутливого контролю та швидкої адаптації рекомендацій. Інтервал 60 хвилин — для зменшення навантаження на API погоди.

Для цілей розробки та тестування передбачено зменшення інтервалів до 1–2 хвилин, що дозволяє оперативно перевіряти реакцію системи на зміну вхідних даних. Початок і завершення кожного завдання логуються з мітками часу. Це дозволяє:

- Переконалися у регулярності виконання завдань.
- Виявити затримки, зависання або збої у планувальнику.
- Аналізувати продуктивність та час виконання окремих етапів

### 2.7.5. Мультитяжіння пристроїв

Архітектура системи розширювана для обслуговування кількох ESP32. Кожен пристрій при надсиланні POST-запиту до `/data` передає у тілі JSON поле `device_id`. Сервер зберігає цей ідентифікатор у четвертому стовпці `sensor_data.csv`, що дозволяє вести облік даних від кількох пристроїв у єдиному сховищі та застосовувати ML-моделі до конкретного набору даних за `device_id`.

Для кожного пристрою можливо зберігати окрему комфортну температуру у вигляді `comfort_temp_<device_id>.txt`, що дозволяє адаптувати рекомендації до умов конкретного приміщення або зони. Запит до `/comfort_temp` може мати параметр `device_id`, за яким сервер повертатиме відповідне значення.

#### 2.7.6. Узгодження з прошивкою ESP32

З метою спрощення серверної логіки відповідальність за відправлення повідомлень у Telegram покладено на ESP32. Це рішення обумовлено наступними міркуваннями:

- ESP32 завжди має актуальний chatID і може самостійно контролювати частоту надсилання повідомлень.
- Сервер не зберігає стан щодо того, коли та кому надсилати сповіщення, що спрощує масштабування.

Опціонально можлива реалізація серверного надсилання повідомлень безпосередньо з `run_analysis`, однак для уникнення спаму передбачається перевірка зміни значення комфортної температури. У поточній реалізації ця функціональність вимкнена за замовчуванням, але описана як можливість для подальшого вдосконалення.

#### 2.8. Інтерфейс користувача та діагностика

Оскільки в поточному прототипі не реалізовано повноцінного веб- чи мобільного інтерфейсу, я зосередився на простих механізмах взаємодії та інструментах діагностики. Цей підрозділ описує, як користувач отримує інформацію, як відбувається діагностика помилок і аналіз даних для оцінки роботи системи.

##### 2.8.1. Telegram-сповіщення як основний канал взаємодії

У межах цього проєкту комунікація між системою та користувачем реалізована через Telegram-бота, що виступає основним каналом передачі інформації. Я створив бота за допомогою BotFather, отримав унікальний токен доступу та вказав ідентифікатор користувача або групи (chatID), куди надсилатимуться повідомлення. Ці параметри були внесені до конфігурації

прошивки мікроконтролера ESP32 і використовуються для встановлення з'єднання з Telegram API.

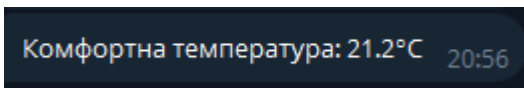


Рис. 2.15 – Приклад повідомлення телеграм бота

Коли система успішно виконує запит до сервера й отримує розраховану модельною частиною комфортну температуру, бот надсилає повідомлення на кшталт: «Рекомендована комфортна температура: X °C», де X — розраховане значення. У випадку, якщо сервер тимчасово недоступний або HTTP-запит повертає код, відмінний від 200 (наприклад, 404 або 500), користувачу надсилається повідомлення про помилку: «Не вдалося отримати рекомендацію від сервера». Окрім цього, у разі, коли сенсор температури й вологості не відповідає протягом кількох циклів опитування, система інформує про це повідомленням типу: «Помилка: датчик температури/вологості не зчитує протягом N циклів».

Щоб уникнути зайвого навантаження на канал зв'язку та не створювати користувачу відчуття спаму, я реалізував механізм контролю частоти сповіщень. Для цього в прошивці ESP32 передбачено порівняння поточного значення рекомендованої температури з попереднім. Якщо нове значення відрізняється менш ніж на 0.5 °C, повідомлення не надсилається. Виняток становлять ситуації, коли мікроконтролер щойно перезапущено і необхідно повідомити початкову рекомендацію.

На поточному етапі Telegram-бот працює в односторонньому режимі, тобто лише приймає повідомлення від пристрою і не обробляє команди від користувача. Проте я передбачив можливість подальшого розвитку цього компонента: за допомогою webhook або періодичного опитування бот може навчитися реагувати на запити користувача, наприклад, на команду відобразити поточні значення температури, вологості або рівень освітлення. Незважаючи на обмежений функціонал, реалізоване рішення є цілком достатнім для демонстрації працездатності системи. Користувач має можливість оперативно

отримувати рекомендації щодо комфортної температури в приміщенні та на основі цього самостійно приймати рішення про зміну налаштувань кліматичної техніки або провітрювання.

### 2.8.2. Перегляд файлів і побудова графіків

Для перевірки накопичених системою даних реалізовано два основні підходи: базовий перегляд та аналітична обробка. Прямий доступ до файлів `sensor_data.csv` та `forecast_hourly.csv` здійснюється через текстовий редактор або табличні процесори, такі як Excel чи LibreOffice Calc. Це дозволяє швидко переконатися в правильності форматування, наявності необхідних полів та коректності даних.

Для глибшого аналізу розроблено окремий скрипт на Python. Він дозволяє зчитувати дані з `sensor_data.csv` у структуру `DataFrame`, перетворювати часові мітки у формат `datetime` та будувати графіки змін температури, вологості та освітленості в часі. Такий візуальний підхід дозволяє оцінити загальну поведінку системи й зміни навколишнього середовища.

Додатково реалізовано накладання історії рекомендованих температур (з файлу `comfort_temp.txt`) на графік фактичної температури в приміщенні. Це дозволяє провести порівняльний аналіз ефективності рекомендацій щодо комфортного мікроклімату, а також виявити відхилення або надмірну реактивність системи.

З метою перевірки якості погодних прогнозів з API здійснюється побудова графіків із `forecast_hourly.csv`, де значення прогнозованої температури порівнюються з фактичними показниками, якщо такі є в архіві. Це дає змогу оцінити достовірність джерела метеоданих і, за потреби, внести корективи в систему ухвалення рішень.

Усі графіки будуються з використанням бібліотеки `matplotlib`. Для кожної фізичної величини створюється окремий графік із підписами осей, легендами та часовою шкалою. У звітній документації рекомендовано включати приклади таких графіків для наочного представлення роботи системи.

Окремий акцент зроблено на валідації моделі прогнозування (LSTM). Якщо в процесі роботи зберігається історія передбачених температур, здійснюється накладання цих даних на графік разом із фактичними. Це дозволяє оцінити точність моделі та, за потреби, коригувати її параметри — наприклад, довжину вхідної послідовності (SEQ\_LEN) або архітектуру нейронної мережі.

### 2.8.3. Логування подій

Для забезпечення контролю за роботою системи передбачено кілька каналів налагодження та виводу діагностичної інформації. З боку мікроконтролера ESP32 використовується серійний порт, що дозволяє через серійний монітор відстежувати ключові повідомлення. Серед них — статус зчитування сенсорів, підключення до Wi-Fi, результати HTTP-запитів до сервера, значення отриманої комфортної температури, а також підтвердження надсилання повідомлень у Telegram. У разі виникнення помилок (наприклад, недоступність сенсора або невдала відповідь від сервера), консоль відображає відповідні попередження, що суттєво прискорює пошук і діагностику проблем.

На стороні серверної частини (реалізованої через Flask) основним інструментом моніторингу слугує вивід у консоль. Логуються всі вхідні запити (зокрема, до маршруту /data), статуси обробки даних, повідомлення про старт серверу. Паралельно працює планувальник APScheduler, який також формує логи із зазначенням часу виконання завдань, успішності функцій `save_next_hour_forecast` та `run_analysis`.

У разі використання модуля машинного навчання для прогнозування температури додатково виводяться метрики якості моделі, зокрема значення MSE або RMSE, а також прогнозовані температури і відповідні комфортні значення. Якщо модель демонструє незадовільну точність, сервер автоматично фіксує факт переходу до резервного (fallback) режиму розрахунків.

У разі, коли сервер використовується також для надсилання Telegram-повідомлень, додатково логуються результати цих відправлень.

Окрім виводу в консоль, передбачено можливість логування в окремі файли. Це може бути реалізовано або через перенаправлення потоку виводу

(stdout) у файл, або з використанням стандартного модуля Python logging, з налаштуванням ротації та збереженням історії. Хоча в поточній тестовій реалізації консолі достатньо для налагодження, можливість розширення через файлові логи зазначається як опціональна для подальшого масштабування системи.

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		52

### 3. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ

У цьому розділі наведено опис підходів до перевірки функціональності окремих компонентів системи, а також тестування їх взаємодії в рамках інтеграційного середовища. Розглянуто методику оцінки ефективності прийнятих рішень, проведено аналіз отриманих результатів і сформульовано висновки щодо переваг та поточних обмежень реалізованого прототипу.

#### 3.1. Тестування окремих компонентів

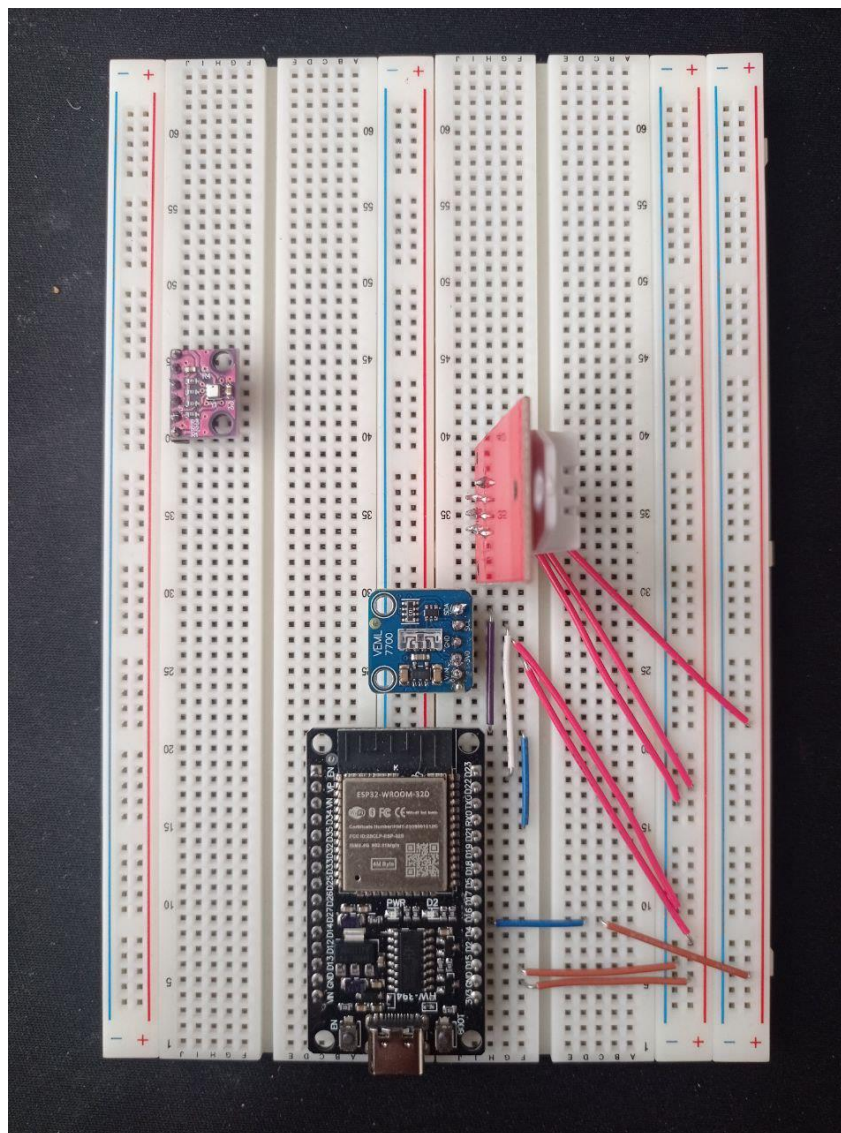


Рис. 3.1 Макетна плата і її компоненти

##### 3.1.1. Тестування ESP32 (зчитування сенсорів і комунікація)

Для забезпечення достовірності отриманих даних було проведено тестування датчиків температури, вологості та освітленості, підключених до

Змін.	Арк.	№ докум.	Підпис	Дата

мікроконтролера ESP32. Зокрема, досліджувалося коректне зчитування показників з сенсорів DHT22 (або BME280) і VEML7700.

Тестування датчиків температури і вологості полягало у порівнянні отриманих показників з контрольними вимірюваннями, здійсненими за допомогою еталонних термометра та гігрометра. Виміри проводились у різних умовах: при підвищеній температурі (поблизу джерела тепла), при охолодженні (відкритому вікні), а також за змінної вологості (біля зволожувача або осушувача повітря). Результати фіксувалися у серійному моніторі ESP32, після чого здійснювався їх аналіз для виявлення систематичних зсувів. За потреби виконувалася калібрувальна корекція програмного забезпечення або перевірка електричних з'єднань.

Для датчика освітленості VEML7700 проводилася перевірка реакції на зміну рівня світла шляхом прикриття сенсора тканиною або навмисним освітленням ліхтарем. Аналізувалася кореляція змін вимірюваного рівня люкс з реальними умовами освітленості для підтвердження адекватності показників.

#### Тестування обробки помилок сенсорів

З метою перевірки стабільності роботи системи у випадку збоїв датчиків було імітовано втрату зв'язку з сенсорами шляхом тимчасового відключення живлення або фізичного розмикання контактів DATA. Відповідна логіка програмного забезпечення повинна була виявляти некоректні значення (наприклад, isnan) або інші помилки зчитування, виконувати логування таких подій та надсилати сповіщення через Telegram після кількох невдалих спроб. Проводилася перевірка коректного відновлення роботи датчиків після повторного підключення.

#### Тестування мережевої взаємодії

Було опрацьовано кілька сценаріїв відправки даних на сервер:

1. Нормальний режим роботи: ESP32 підключається до Wi-Fi, сервер доступний. Дані періодично відправляються методом POST на endpoint /data. Перевірялось коректне записування отриманих даних у файл sensor\_data.csv на сервері.

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		54

2. Втрата Wi-Fi з'єднання: імітувалось вимкнення точки доступу або зміна пароля. Програмне забезпечення ESP32 повинно було виявити відсутність мережі, здійснювати повторні спроби підключення, при цьому кешувати або зберігати у внутрішній пам'яті останні вимірні значення, а після відновлення з'єднання — відправляти накопичені дані. При цьому події логувалися, а у разі передбачення — надсилалися повідомлення у Telegram.
3. Недоступність сервера: при наявності Wi-Fi, але відсутності відповіді від сервера (вимкнений сервер або змінена IP-адреса), ESP32 робила спроби підключення, отримувала помилки з'єднання або HTTP-коди, відмінні від 200. У цих випадках відбувалося логування помилок, періодичне повторення запитів та, за наявності відповідного функціоналу, надсилання Telegram-сповіщень про проблеми зв'язку.

#### Тестування отримання рекомендації

Після успішної відправки даних методом POST виконувався GET-запит на endpoint /comfort\_temp для отримання рекомендованого комфортного значення температури. Перевірялися наступні випадки:

1. Сервер повертає коректне значення (наприклад, "22.5"), яке ESP32 конвертує у числовий формат (float) та зберігає для подальшої обробки.
2. Сервер повертає некоректне значення або внутрішню помилку (код 500). У цьому випадку ESP32 здійснює логування помилки та, відповідно до налаштувань, надсилає повідомлення Telegram про неможливість отримати рекомендацію.
3. Сервер повертає значення, що виходить за межі допустимого діапазону. Система виявляє аномалію через перевірку діапазону або функцію isnan, виконує логування та не використовує такі некоректні дані.

#### Тестування Telegram-сповіщень

Для перевірки функціональності надсилання повідомлень через Telegram API проводилися наступні тести:

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		55

1. У нормальному режимі ESP32 успішно викликала API Telegram, отримувала відповідь з кодом 200, що підтверджує доставку повідомлення.
2. Імітація недоступності Telegram API (наприклад, зміна URL або відсутність інтернет-з'єднання). У такому разі ESP32 виявляє помилку, виконує логування, а за реалізацією — здійснює повторні спроби або повідомляє про помилку.
3. Перевірка обмежень частоти надсилання повідомлень. Повідомлення відправляються лише при зміні рекомендованої комфортної температури на величину, що перевищує заданий поріг (наприклад, 0.5 °C), або при перших запуску пристрою чи фіксації помилки. Це дозволяє уникнути надмірної кількості сповіщень та потенційного спаму.

### 3.1.2. Тестування серверної частини (Flask)

Для перевірки коректності роботи маршрутів веб-сервера розроблено набір юніт-тестів із використанням Flask test client. Тестування охоплює такі сценарії:

1. При надходженні POST-запиту на маршрут /data із валідним JSON-форматом очікується відповідь із кодом 200. Дані зберігаються у тимчасовому CSV-файлі, створеному спеціально для тестування. Перевіряється наявність нового рядка та відповідність його формату заданому шаблону.

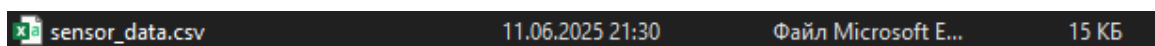


Рис. 3.2 Створення csv фалу при першому запуску

2. Якщо POST-запит містить неповний або некоректний JSON (наприклад, відсутнє обов'язкове поле або значення є нечисловим), сервер повинен повернути код 400, а запис у файл не здійснюється.

	A
1	2025-06-08 02:35:33,28.6,54.2,159.67
2	2025-06-08 02:37:43,28.6,53.9,159.32
3	2025-06-08 02:41:36,28.6,54.3,159.32
4	2025-06-08 02:41:52,28.5,54.3,159.38
5	2025-06-08 02:42:52,28.5,54.5,158.8
6	2025-06-08 02:43:53,28.4,55.2,16.07
7	2025-06-08 03:07:02,28.6,54.8,159.84
8	2025-06-08 14:26:21,31.7,49.0,206.38
9	2025-06-08 14:27:21,31.5,48.8,561.6
10	2025-06-08 14:28:21,31.5,48.9,554.86
11	2025-06-08 14:29:21,31.4,49.4,549.16
12	2025-06-08 14:30:22,31.4,49.8,227.64
13	2025-06-08 14:31:22,31.4,49.8,206.15
14	2025-06-08 14:32:22,31.5,48.9,193.71
15	2025-06-08 14:33:22,31.7,48.8,211.33
16	2025-06-08 14:34:23,31.7,49.3,188.81
17	2025-06-08 14:35:23,31.8,48.8,201.14
18	2025-06-08 14:36:23,31.8,47.4,198.37
19	2025-06-08 14:37:23,31.8,47.4,223.26
20	2025-06-08 14:38:24,31.7,47.7,213.93

Рис. 3.3 Створення csv файлу при першому запуску

3. При GET-запиті до маршруту /comfort\_temp, якщо файл comfort\_temp.txt містить валідне числове значення, сервер повертає цей рядок із кодом 200.
4. Якщо файл comfort\_temp.txt відсутній або містить некоректні дані, сервер повертає код 500 та записує відповідне повідомлення у лог.

```
Відповідь сервера: 400
Отримано комфортну температуру: 21.2
Повідомлення надіслано в Telegram.
```

Рис. 3.4 Тестування роботи запитів від ESP32

### 3.1.3. Тестування модуля прогнозу погоди

Для забезпечення коректної роботи модуля прогнозування погоди було проведено низку тестів, які охоплюють перевірку парсингу API, обробку мережових помилок, логування та інтервали виклику.

#### Перевірка коректного парсингу API

1. Функція `get_next_hour_forecast()` тестувалась із реальним запитом до погодного API. Перевірялось, що результатом виклику є кортеж із двох значень — часової позначки прогнозу (`forecast_time`) та температури

(temperature), формат яких відповідає очікуванням сервера та подальшої обробки.

2. Імітувалась некоректна відповідь API, наприклад, шляхом змінення ключів у отриманому JSON-об'єкті. У такому випадку перевірялось, що код коректно ловить виняток KeyError або реагує на відсутність очікуваних полів. Функція в таких ситуаціях повертає кортеж (None, None) і записує відповідне повідомлення у лог.

#### Тестування поведінки в мережі

1. Для перевірки стійкості до відсутності інтернет-з'єднання тестувалось відключення мережі. Перевірялось, що запит до API коректно таймаутиться, виключення ловиться, а функція повертає (None, None) без аварійного завершення.
2. Встановлювався короткий таймаут HTTP-запиту для імітації повільної мережі. Оцінювалась поведінка модуля при затримках, що перевищують встановлені межі, з метою перевірки стабільності та коректного оброблення помилок.

#### Логування історії прогнозів

При кожному виклику функції `save_next_hour_forecast()` перевіряється додавання нового рядка до журналу прогнозів. Рядок повинен містити коректні часові позначки запиту і прогнозу, а також значення температури. Аналізу піддавались кілька записів для підтвердження узгодженості формату.

#### Перевірка інтервалів виклику

В ході розробки інтервал виклику прогнозу встановлювався коротшим (наприклад, 5 хвилин) для оперативного тестування. У фінальній конфігурації інтервал встановлено на 30 хвилин з метою уникнення перевантаження API. Логування дозволяє відстежувати зміну прогнозних значень у часі: якщо температура для наступної години відрізняється від попереднього запису, у лог додається новий рядок із актуальними даними.

```

Scheduler запущено: tasks every 30 minutes
* Serving Flask app 'sensor'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.108:5000
Press CTRL+C to quit

```

Рис 3.5 Приклад запуску сервера і тестування кожні 30 хвилин

### 3.1.4. Тестування ML-модуля

Для перевірки коректності роботи моделі прогнозування температури проведено низку тестів, які охоплюють підготовку даних, роботу з різними обсягами вхідної інформації, тренування та оцінку як на синтетичних, так і на реальних даних, а також обробку виключень.

#### Підготовка тестових даних

Для тестування моделі створено набір синтетичних даних у файлі `sensor_data.csv`. Дані генеруються у вигляді послідовності температур із добовим циклом — наприклад, синусоїда з додаванням випадкового шуму. Крім того, додаються випадкові значення вологості та освітленості. Такий набір дозволяє перевірити, чи LSTM-модель виявляє кореляції між параметрами та адекватно прогнозує продовження тренду.

#### Тестування на недостатній кількості даних

Модель запускалась із дуже малою історією даних (3–4 записи). Перевірялось, що код виявляє недостатність інформації для прогнозування, логуючи відповідне повідомлення, та застосовує `fallback` — повертає останнє відоме значення температури.

#### Тренування та оцінка на штучному наборі даних

Для оцінки працездатності моделі використовується штучний набір з кількох сотень точок із відомою закономірністю. Перевіряється, що LSTM тренується без помилок, а прогнозовані значення близькі до очікуваних. Оцінка якості проводиться за метриками MSE (Mean Squared Error) і RMSE (Root Mean Squared Error). Отримані результати порівнюються з вихідною формулою генерації даних для підтвердження адекватності.

## Тестування на реальних даних

За наявності реальних записів внутрішньої температури запускається прогноз на цих даних. Порівнюється прогноз з фактичним наступним вимірюванням, аналізується похибка. На основі цього приймається рішення щодо необхідності коригування архітектури моделі: зміни довжини послідовності (SEQ\_LEN), кількості нейронів або числа епох тренування.

## Обробка виключень під час тренування

Імітуються ситуації виникнення помилок під час навчання моделі, наприклад через відсутність пам'яті або пошкодження файлу збереженої моделі. Перевіряється, що блок try/except коректно ловить виключення, повідомлення про помилку записується у лог, і далі застосовується fallback-прогноз.

## Тестування альтернативної моделі

Для порівняння точності іноді запускається модель RandomForest. Для неї формуються ознаки: температура, вологість, освітленість, година доби, сезон, а ціль — наступне значення температури. Перевіряється, чи RandomForest дає порівнянну або кращу точність за умов малого обсягу даних. Результати логуються, а в подальшому аналізі використовується найбільш ефективний підхід.

```
C:\My\KPI\Diploma\ESP32_Scan_Sensor>python climate_analyzer.py
2025-06-12 10:36:15.145373: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-06-12 10:36:16.541679: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.

[2025-06-12 10:36:19] Запуск run_analysis()
[RandomForest] RMSE: 0.1592
2025-06-12 10:36:19.383118: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[Loaded LSTM model from lstm_model.keras
[LSTM] Test Loss (MSE): 0.0836
[LSTM] Forecasted next temperature: 27.82 °C
[Hybrid Comfort Model] Final comfort recommendation: 21.4 °C
[Комфортна температура збережена у файл: 21.4 °C
```

Рис 3.6 Робота нейронної мережі LSTM

### 3.1.5. Тестування алгоритму розрахунку комфортної температури

Для забезпечення коректної роботи функції calculate\_comfort\_temperature було розроблено як ручні, так і автоматизовані сценарії тестування.

## Ручні сценарії

Створено набір тестових випадків із різними комбінаціями вхідних параметрів:

1. Температура всередині 20 °С, вологість 80%, низька освітленість, зовнішній прогноз — теплий

Очікувано зниження рекомендованої температури в межах допустимого діапазону.

2. Температура всередині 24 °С, вологість 30%, висока освітленість, зовнішній прогноз — холодний

Очікується підвищення або мінімальна корекція комфортної температури.

3. Граничні умови: холодно всередині, висока вологість, зовні холодно  
Рекомендація комфортної температури повинна залишатися в межах максимально допустимого діапазону (19–25 °С). Перевіряється, що алгоритм не виходить за ці межі.

Для кожного сценарію виконувався виклик функції `calculate_comfort_temperature` з відповідними параметрами. Результати порівнювались із очікуваними значеннями, базованими на логіці алгоритму. У разі невідповідностей пороги та ваги корекції переглядалися і коригувалися.

Також розроблено набір юніт-тестів, які передають у функцію різні комбінації вхідних параметрів, перевіряють, що функція повертає значення в очікуваному діапазоні, контролюють напрямок корекції (підвищення чи зниження температури), щоб гарантувати відповідність бізнес-логіці.

Ці автоматизовані тести забезпечують регресійну перевірку — при внесенні змін до логіки алгоритму базові правила залишаються незмінними.

### 3.1.6. Тестування Telegram-бота (сервер/ESP32)

Для перевірки роботи модуля надсилання повідомлень через Telegram-бота було проведено низку тестів, спрямованих на оцінку як змісту повідомлення, так і стійкості системи до помилок. Найперше перевірялася коректність сформульованого тексту повідомлення. Повідомлення типу “Рекомендована комфортна температура: X °С” повинно бути чітким, зрозумілим, відповідати нормам української мови та не містити зайвих або неоднозначних формулювань.

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		61

Особливу увагу приділено локалізації — правильному вживанню одиниць виміру та граматичній узгодженості.

Окремо тестувалась стійкість модуля до перебоїв із мережею. У разі вимкнення або втрати інтернет-з'єднання під час надсилання повідомлення перевірялося, що система не аварійно завершується. Повідомлення або ігноруються, або кешуються, залежно від реалізації, а в логах фіксується відповідне попередження. Це дозволяє зберігати працездатність навіть у нестабільних умовах.

Також було змодельовано типові помилки Telegram API, зокрема некоректні значення токена (botToken) або ідентифікатора чату (chatID). У таких випадках функція повинна перехоплювати винятки і фіксувати в логах зміст помилки, наприклад повідомлення на кшталт “Unauthorized” або опис відповідної відповіді від API. Такий підхід дозволяє оперативно виявляти проблеми конфігурації або доступу й уникати прихованих збоїв у роботі системи.

```
// --- Налаштування Wi-Fi ---  
const char* ssid = "300 bucks";  
const char* password = "12345678";  
const char* serverURL = "http://192.168.0.108:5000/data"; // IP ПК  
// --- Telegram ---  
String botToken = "7550609346:AAG8_A3uQeDV16wBdIpDRS2UTEFsItm5_0M";  
String chatID = "636826226";
```

Рис. 3.7 Задання параметрів мережі та телеграм бота

### 3.2. Інтеграційні тести

Для перевірки взаємодії всіх компонентів системи проведено повноцінне інтеграційне тестування. Насамперед перевірявся повний end-to-end сценарій, який охоплює повний цикл: мікроконтролер ESP32 зчитує дані (реальні або емуляторні) та надсилає їх на сервер через POST /data. Сервер приймає ці дані та записує у файл sensor\_data.csv. Далі згідно з розкладом APScheduler викликає функцію save\_next\_hour\_forecast, яка формує прогноз погоди, а потім — run\_analysis, що проводить прогноз внутрішньої температури, розраховує рекомендовану комфортну температуру та зберігає її у файл comfort\_temp.txt.

ESP32 надсилає запит GET /comfort\_temp, отримує нове значення і надсилає повідомлення користувачу через Telegram. Кінцевий користувач отримує рекомендацію і може на її основі вручну налаштувати кліматичне обладнання в будинку.

```
Підключення до WiFi...
Підключення до WiFi...
Підключення до WiFi...
WiFi підключено
IP адреса ESP32: 192.168.0.101
JSON: {"temperature":24.20,"humidity":49.80,"light":1985}
Відповідь сервера: -1
Не вдалося отримати комфортну температуру. Код: -1
Повідомлення надіслано в Telegram.
JSON: {"temperature":24.20,"humidity":49.70,"light":784}
Відповідь сервера: -1
Не вдалося отримати комфортну температуру. Код: -1
Повідомлення надіслано в Telegram.
JSON: {"temperature":24.10,"humidity":49.90,"light":1653}
Відповідь сервера: 400
Отримано комфортну температуру: 21.2
Повідомлення надіслано в Telegram.
JSON: {"temperature":24.00,"humidity":49.70,"light":754}
Відповідь сервера: 400
Отримано комфортну температуру: 21.2
Повідомлення надіслано в Telegram.
```

Рис.3.8 Повний цикл роботи ESP32

Під час тестового розгортання я ретельно відстежував кожен етап цього ланцюжка в логах, щоб переконатися, що передача даних відбувається своєчасно, інформація не губиться, а функціонування системи є стабільним.

Додатково було протестовано зміну зовнішніх і внутрішніх умов. Зокрема, моделювалась зміна прогнозу погоди як через зміну відповіді API, так і в реальному часі, щоб відстежити вплив на рекомендації щодо температури.

Також перевірялась реакція системи на коливання внутрішньої температури — приміщення штучно нагрівалося або охолоджувалося. Крім того, змінювався рівень вологості (через зволоження або осушення), щоб переконатися, що система правильно враховує ці фактори в аналітиці.

Окрему увагу приділено стійкості системи до помилок та нестандартних ситуацій. При імітації втрати зв'язку під час надсилання POST перевірялося, що ESP32 кешує дані локально і надсилає їх повторно після відновлення з'єднання,

при цьому сервер отримує лише повні рядки, без часткових чи пошкоджених. У випадку, коли у файлі `sensor_data.csv` занадто мало записів, система коректно переходить до резервного режиму без використання машинного навчання, про що повідомляє в логах. Некоректні або пошкоджені рядки в цьому ж файлі обробляються стійко — вони ігноруються або видаляються, що не призводить до аварійного завершення роботи.

Симульовані також збої при отриманні прогнозу погоди — у разі помилки API функція `save_next_hour_forecast` ловить виключення, а `run_analysis` переходить у режим без прогнозу, використовуючи попередні або базові значення. Перевірялась також ситуація з одночасним доступом до файлу `sensor_data.csv`, коли можливі конфлікти між читанням і записом з різних потоків. У таких випадках було додано простий механізм блокування (через `threading.Lock`), що забезпечує цілісність даних.

Щодо періодичних задач, було перевірено, що виклики `APScheduler` залишаються стійкими навіть у разі помилок усередині задач. Наприклад, у функції `run_analysis` навмисно вводилась помилка (ділення на нуль або звернення до неіснуючої змінної), після чого переконанося, що ця ситуація обробляється, логуються повідомлення, а наступне виконання задачі відбувається в звичному режимі. Також перевірявся час виконання завдань — якщо попередня задача ще виконується, нова не запускається паралельно (через параметри `coalesce` або інші обмеження). Додатково моніторились споживання ресурсів системи під час запуску `run_analysis` — як процесора, так і оперативної пам'яті, зокрема на Raspberry Pi. У разі надмірного навантаження передбачено зниження частоти запуску або обсягів обробки.

### 3.3. Оцінка потенційної ефективності

Для оцінки потенційної ефективності системи без автоматичного керування реле було проведено моделювання на основі зібраних даних про внутрішню температуру, зовнішню температуру (або прогноз погоди) та умовну активність кліматичного обладнання. Метою було наближене визначення можливої економії енергії та впливу на комфорт у приміщенні.

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		64

Модельна оцінка базувалась на порівнянні двох сценаріїв: підтримання фіксованої температури (наприклад, 22 °С) без урахування погодних умов — як це зазвичай відбувається при традиційному керуванні — та режиму, у якому користувач слідує рекомендованим значенням температури, що враховують зовнішні умови. Для оцінки втрат тепла використовувалась спрощена формула теплового потоку

$Q = k * \Delta T$ , де  $\Delta T$  — різниця між внутрішньою та зовнішньою температурою, а  $k$  — умовна стала, що відповідає характеристикам приміщення. В адаптивному режимі, завдяки рекомендаціям системи, передбачалося зменшення середньої температурної різниці, що призводить до зниження енерговитрат.

Додатково враховувалися добові коливання температури: система пропонує знижувати температуру перед нічним похолоданням або підвищувати її перед ранковим охолодженням, що зменшує навантаження на обладнання в критичні періоди. Результати моделювання показали, що за середнього місячного споживання на рівні 1000 кВт·год можлива економія становить 10–20% за умови дотримання рекомендацій. Точна величина залежить від теплотехнічних властивостей будівлі, типу обладнання та поведінки користувача.

Слід зазначити, що модель є наближеною і не враховує інерційність конструкцій, вплив сонячного випромінювання, втрати через вентиляцію та інші чинники. Крім того, ефективність залежить від ступеня дотримання рекомендацій: у разі їх ігнорування економія не досягається. Для точного визначення впливу системи необхідне встановлення вимірювачів споживання енергії.

Щодо аналізу комфортності, оцінювались коливання внутрішньої температури навколо бажаного рівня. Як метрику використано стандартне відхилення температурних значень. У звичайному режимі спостерігались коливання в межах  $\pm 2-3$  °С, тоді як у разі реагування на рекомендації спостерігалось зменшення розмаху до  $\pm 0.5-1$  °С. Аналіз проводився на основі

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		65

записів у файлі sensor\_data.csv: порівнювались періоди до та після активного впровадження рекомендацій. Наприклад, стандартне відхилення могло знизитися з 1.5 °C до 0.8 °C, що свідчить про стабільніший мікроклімат у приміщенні.

Втім, ефект залежить від своєчасності реакції користувача на поради. При запізній зміні температури очікувана стабільність може не реалізуватися. Крім того, на мікроклімат впливають інші фактори, зокрема інсоляція та робота окремих пристроїв.

Також враховувався якісний зворотній зв'язок. Аналізувалися враження від отримання рекомендацій у Telegram, зокрема зручність формулювань, інтервал надсилання повідомлень, потреба в додатковій інформації (наприклад, очікувана економія чи графіки). Ці спостереження є неформальними, однак допомагають сформулювати уявлення про практичну доцільність системи та визначити напрями покращення інтерфейсу взаємодії з користувачем.

## ВИСНОВКИ

У ході реалізації дипломного проєкту було створено прототип інтелектуальної системи енергоменеджменту для «розумного будинку» з використанням контролера ESP32, базових сенсорів (температури, вологості, освітленості), файлового сховища та серверної частини на Flask з модулем прогнозу погоди та ML-модулем для прогнозування внутрішньої температури.

Основною метою було продемонструвати можливість адаптивного підходу до керування мікрокліматом без потреби у складній інфраструктурі. Поставлена мета була досягнута: система здійснює збір даних, прогнозує майбутні значення температури, розраховує рекомендації щодо комфортної температури за гібридним алгоритмом і надсилає їх користувачеві через Telegram.

У процесі тестування підтверджено стабільність роботи прошивки ESP32: здійснюється надійне зчитування та передача показників, реалізована обробка помилок сенсорів і збоїв зв'язку. Серверна частина на Flask коректно приймає дані, регулярно отримує прогноз погоди та виконує ML-аналіз. У разі обмеженого обсягу історичних даних застосовується fallback-прогноз. Модель LSTM забезпечує адекватні результати прогнозування за умови наявності достатнього обсягу вхідних даних, а за нестачі — система переходить до використання останнього зафіксованого значення температури. Алгоритм розрахунку комфортної температури діє згідно з визначеними правилами, обмежуючи результати у межах безпечного діапазону.

Інтеграція модуля APScheduler забезпечує періодичне оновлення прогнозів і рекомендацій. Механізм сповіщення через Telegram гарантує інформування користувача. Проведені інтеграційні тести засвідчили стійкість системи до типових збоїв, зокрема втрати мережі, недоступності API прогнозу погоди або пошкодження даних. Завдяки реалізованій обробці винятків і fallback-логіці система зберігає працездатність навіть за несприятливих умов.

Разом із тим, прототип має низку обмежень. Відсутність автоматичного керування кліматичним обладнанням вимагає участі користувача для реалізації рекомендацій. Файлове сховище обмежує масштабованість системи та може

					ІАЛЦ.467200.004 ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		67

ускладнювати довготривалу експлуатацію без впровадження ротації або переходу на СУБД. Інтерфейс користувача реалізовано лише у вигляді Telegram-сповіщень, що передбачає ручний аналіз зібраної інформації. ML-модель вимагає накопичення даних для підвищення точності прогнозів.

У якості напрямів подальшого розвитку розглядається підключення реле для автоматизації вмикання/вимикання кліматичного обладнання із дотриманням вимог безпеки та гістерезису; перехід на використання бази даних для зберігання великих обсягів інформації та її ефективної обробки; створення веб- або мобільного інтерфейсу з візуалізацією графіків і можливістю налаштувань; розширення сенсорної частини за рахунок детектора присутності, сенсора CO<sub>2</sub> тощо; удосконалення ML-модуля шляхом впровадження альтернативних алгоритмів, оптимізації гіперпараметрів та реалізації онлайн-навчання.

Розробка та реалізація даного прототипу надали цінний практичний досвід у галузі апаратної інтеграції, обробки мережевих збоїв, побудови моделей машинного навчання та організації періодичних задач. Створена система довела, що концепція адаптивного клімат-контролю з використанням прогнозів і рекомендацій може бути реалізована навіть на відносно простому апаратному забезпеченні. Незважаючи на відсутність повної автоматизації, прототип здатен виступати як ефективний порадник для зниження енергоспоживання та підвищення рівня комфорту в умовах реального використання.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ESP32 [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.espressif.com/en/products/socs/esp32>
2. Arduino IDE [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.arduino.cc/en/software>
3. DHT22 (датчик температури й вологості) [Електронний ресурс]. – Режим доступу до ресурсу: <https://learn.adafruit.com/dht/overview>
4. BME280 (датчик температури, вологості та тиску) [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/>
5. VEMML7700 (Adafruit\_VEMML7700, датчик освітленості) [Електронний ресурс]. – Режим доступу до ресурсу: <https://learn.adafruit.com/adafruit-vemml7700-lux-sensor-breakout/>
6. Flask [Електронний ресурс]. – Режим доступу до ресурсу: <https://flask.palletsprojects.com/>
7. APScheduler [Електронний ресурс]. – Режим доступу до ресурсу: <https://apscheduler.readthedocs.io/>
8. pandas [Електронний ресурс]. – Режим доступу до ресурсу: <https://pandas.pydata.org/>
9. NumPy [Електронний ресурс]. – Режим доступу до ресурсу: <https://numpy.org/>
10. TensorFlow і Keras (LSTM) [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.tensorflow.org/guide/keras/rnn>
11. scikit-learn [Електронний ресурс]. – Режим доступу до ресурсу: <https://scikit-learn.org/>
12. sklearn.preprocessing.MinMaxScaler [Електронний ресурс]. – Режим доступу до ресурсу: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>
13. requests (Python) [Електронний ресурс]. – Режим доступу до ресурсу: <https://docs.python-requests.org/>

14. OpenWeatherMap API [Електронний ресурс]. – Режим доступу до ресурсу: <https://openweathermap.org/api>
15. Open-Meteo API [Електронний ресурс]. – Режим доступу до ресурсу: <https://open-meteo.com/>
16. Telegram Bot API [Електронний ресурс]. – Режим доступу до ресурсу: <https://core.telegram.org/bots/api>
17. MQTT (Message Queuing Telemetry Transport) [Електронний ресурс]. – Режим доступу до ресурсу: <https://mqtt.org/>
18. Model predictive control of indoor microclimate: existing building stock comfort improvement [Електронний ресурс]. – Режим доступу до ресурсу: <https://arxiv.org/abs/1806.08999>
19. Adaptive Control of Building HVAC Systems Using Reinforcement Learning [Електронний ресурс]. – Режим доступу до ресурсу: [https://www.researchgate.net/publication/351327436\\_Adaptive\\_Control\\_of\\_Building\\_HVAC\\_Systems\\_Using\\_Reinforcement\\_Learning](https://www.researchgate.net/publication/351327436_Adaptive_Control_of_Building_HVAC_Systems_Using_Reinforcement_Learning)
20. Weather-Responsive Building Energy Management Using Machine Learning [Електронний ресурс]. – Режим доступу до ресурсу: <https://link.springer.com/article/10.1007/s12053-019-09785-0>
21. Raspberry Pi (для розгортання серверної частини) [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.raspberrypi.org/>
22. Git та GitHub (версіонування) [Електронний ресурс]. – Режим доступу до ресурсу: <https://git-scm.com/>; <https://github.com/>
23. Python [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.python.org/>
24. ESP32 Arduino Core [Електронний ресурс]. – Режим доступу до ресурсу: <https://github.com/espressif/arduino-esp32>
25. Adafruit Unified Sensor Library [Електронний ресурс]. – Режим доступу до ресурсу: [https://github.com/adafruit/Adafruit\\_Sensor](https://github.com/adafruit/Adafruit_Sensor)

