Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Кафедра комп'ютерних систем та робототехніки

До захисту допущено

Кафедрою комп'ютерних систем та робототехніки

протокол № __ від __ червня 2026 р.

завідувач кафедри _____ Максим ХРУСЛОВ

«___»_____2026 р.

Кваліфікаційна робота

здобувача першого (бакалаврського) рівня вищої освіти

«МОДЕЛЬ СИСТЕМИ АВТОМАТИЧНОГО УПРАВЛІННЯ ПРОЦЕСОРАМИ НА БАЗІ ESP32 З ЗАСТОСУВАННЯМ ЕЛЕМЕНТІВ ШТУЧНОГО ІНТЕЛЕКТУ»

Спеціальність ***151 – Автоматизація та комп'ютерно-інтегровані технології***

Освітня програма ***Автоматизація та комп'ютерно інтегровані технології***

Виконавець _____ Олексій НИКОЛАЄНКО

Науковий керівник _____ Олександр РАЛО

Харків - 2026

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра присвячена розробці моделі системи автоматичного управління процесорами на базі мікроконтролера ESP32 із застосуванням елементів штучного інтелекту. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи складає 63 сторінки, із яких 32 сторінок основної частини з 4 рисунками, 2 таблицями, 8 найменувань у списку використаних джерел та шість додатків.

Метою кваліфікаційної роботи є підвищення ефективності систем автоматичного управління процесорами шляхом застосування нейромережевої моделі прийняття рішень, реалізованої на базі мікроконтролера ESP32.

Об'єктом дослідження є процеси автоматичного управління процесорами та прийняття керуючих рішень на базі мікроконтролера ESP32.

Предметом дослідження є мікроконтролерні методи, алгоритми та моделі побудови систем автоматичного управління процесорами на базі мікроконтролера ESP32 з використанням штучних нейронних мереж архітектури MLP.

У роботі досліджено архітектуру ESP32, розроблено модель системи автоматичного управління на основі MLP, проведено моделювання її роботи та аналіз використання ресурсів мікроконтролера

Ключові слова: автоматизація та приладобудування, система автоматичного управління, штучний інтелект, нейронна мережа, ESP32, мікроконтролер, інтелектуальне управління.

ABSTRACT

The explanatory note to the Bachelor's qualification thesis is devoted to the development of a model of an automatic processor control system based on the ESP32 microcontroller using artificial intelligence elements. The thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices. The total volume of the thesis is 63 pages, including 32 pages of the main body, 4 figures, 2 tables, 8 references, and 6 appendices.

The aim of the qualification thesis is to improve the efficiency of automatic processor control systems through the application of a neural network decision-making model implemented on the ESP32 microcontroller platform.

The object of research is the processes of automatic processor control and decision-making based on the ESP32 microcontroller.

The subject of research is microcontroller-based methods, algorithms, and models for designing automatic processor control systems using the ESP32 microcontroller and MLP-based artificial neural networks.

The thesis investigates the ESP32 architecture, develops an MLP-based automatic control system model, performs simulation of its operation, and analyzes the utilization of microcontroller resources.

Keywords: automation and instrumentation, automatic control system, artificial intelligence, neural network, ESP32, microcontroller, intelligent control.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ 6
ВСТУП 7
РОЗДІЛ 1 АНАЛІЗ ТА ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ САУ НА БАЗІ ESP32 З ЕЛЕМЕНТАМИ ШІ 9
1.1 Архітектурні особливості та технічні характеристики ESP32 9
1.2. Аналіз сучасних підходів до побудови системи автоматичного управління 11
1.3. Методи побудови моделей систем автоматичного управління 12
1.4. Інтеграція елементів штучного інтелекту у вбудовані системи 13
1.5. Аналіз алгоритмів ШІ 15
Висновки за розділом 1 17
РОЗДІЛ 2 ПРОЄКТУВАННЯ МОДЕЛІ СИСТЕМИ АВТОМАТИЧНОГО УПРАВЛІННЯ ПРОЦЕСОРАМИ 18
2.1. Концептуальна модель 18
2.2. Розробка структури та алгоритму датасету 21
2.2.1. Структура та синтез набору даних 21
2.2.2. Алгоритм нормалізації даних 22
2.3. Програмна реалізація 23
2.3.1. Головний цикл управління 23
2.3.2. Модуль нейромережі 25
Висновки за розділом 2 27
РОЗДІЛ 3 МОДЕЛЮВАННЯ, ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ 28
3.1. Інструментарій розробки програмного забезпечення та середовище симуляції 28
3.2 Результати моделювання 29
Висновки за розділом 3 32
ВИСНОВКИ 34

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 35
ДОДАТКИ 36

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

ШІ – Штучний інтелект

САУ – Система автоматичного управління

MLP – Multilayer Perceptron (Багатошаровий персептрон)

ВСТУП

Актуальність роботи. Сучасний етап розвитку систем автоматичного керування характеризується переходом від класичних жорстких алгоритмів регулювання до інтелектуальних систем, адже підходи, які базуються на жорстких логічних умовах типу «if-then», часто виявляються неефективними під час роботи в умовах невизначеності, зашумленості даних або за наявності складних нелінійних взаємозв'язків між параметрами середовища, але з іншого боку, бурхливий розвиток концепції штучного інтелекту на вбудованих системах відкрив можливість перенести обчислення нейромережевих моделей безпосередньо на мікроконтролери і при цьому без необхідності постійного підключення до потужних хмарних серверів, що знижує трафік, зменшує час відгуку системи до мікросекунд і забезпечує повну автономність роботи.

Мета дослідження – підвищення ефективності систем автоматичного управління процесорами шляхом застосування нейромережевої моделі прийняття рішень, реалізованої на базі мікроконтролера ESP32.

Об'єкт дослідження – процеси автоматичного управління процесорами та прийняття керуючих рішень на базі мікроконтролера ESP32.

Предмет дослідження – мікроконтролерні методи, алгоритми та моделі побудови систем автоматичного управління процесорами на базі мікроконтролера ESP32 з використанням штучних нейронних мереж архітектури MLP.

Методи дослідження: методи системного аналізу, теорія автоматичного управління, теорія штучних нейронних мереж, методи математичного моделювання, методи об'єктно-орієнтованого програмування.

Завдання дослідження:

1.Проаналізувати підходи до побудови систем автоматичного управління на базі мікроконтролерів та дослідити архітектурні особливості ESP32 для задач керування.

2.Дослідити методи побудови моделей автоматичного управління та визначити можливості інтеграції елементів штучного інтелекту у вбудовані системи.

3.Розробити модель системи автоматичного управління процесорами на базі ESP32 з використанням обраних елементів штучного інтелекту.

4.Розробити структурну схему інтелектуальної системи, детально описати її функціональні модулі та принципи їхньої взаємодії.

5.Провести комп'ютерне моделювання роботи розробленої системи, оцінити якість регулювання та виконати аналіз отриманих результатів.

РОЗДІЛ 1
АНАЛІЗ ТА ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ САУ НА БАЗІ ESP32 З ЕЛЕМЕНТАМИ ШІ

1.1 Архітектурні особливості та технічні характеристики ESP32

ESP32 – це високопродуктивна система-на-кристалі, орієнтована на застосування в системах Інтернету речей (IoT), автоматизації та вбудованих інтелектуальних пристроях, через що він широко використовується при створенні систем автоматичного керування, зокрема з ШІ. Цей мікроконтролер виконано за 40-нм технологією TSMC, яка об'єднує в одному кристалі модулі Wi-Fi, Bluetooth, обчислювальні ядра, пам'ять, а також широкий набір периферійних інтерфейсів [1].

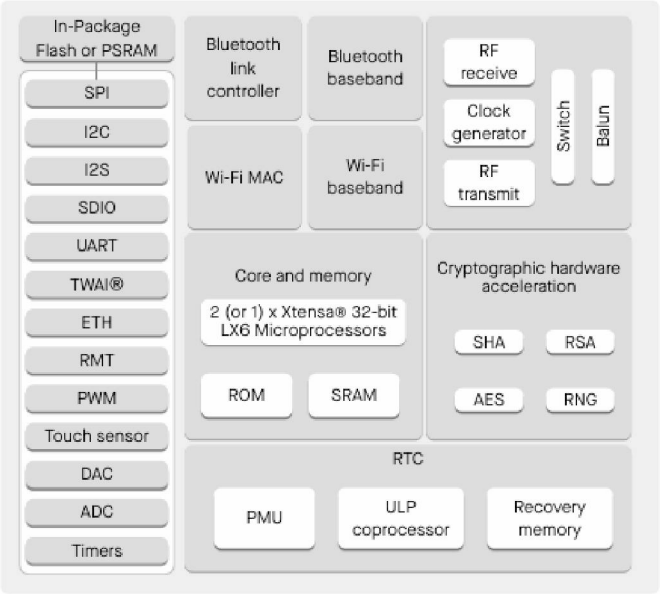


Рисунок 1.1 – Блок-схема мікроконтролера ESP32.

Основою обчислювальної підсистеми є одно- або двоядерні 32-розрядні процесори Xtensa LX6 із тактовою частотою до 240 МГц. Продуктивність двоядерної конфігурації сягає 1079.96 CoreMark, це забезпечує можливість виконання задач цифрової обробки сигналів, адаптивного керування та роботи

нейромережевих моделей безпосередньо на мікроконтролері. Архітектура включає в себе 448 КБ постійної пам'яті ROM, 520 КБ оперативної пам'яті SRAM та 16 КБ RTC SRAM. Підтримується підключення зовнішньої Flash-пам'яті та PSRAM через інтерфейс QSPI, що дає змогу розширювати ресурси пам'яті при реалізації складніших алгоритмів керування та обробки даних. Наявність окремої RTC-пам'яті забезпечує збереження даних у режимах зниженого енергоспоживання. Також ESP32 має вбудований блок бездротового зв'язку. Мікроконтролер підтримує стандарт Wi-Fi IEEE 802.11 b/g/n зі швидкістю передачі даних до 150 Мбіт/с, а також Bluetooth v4.2 і Bluetooth Low Energy, що дає змогу інтегрувати пристрій у розподілені системи автоматизації, віддалено стежити за параметрами та передавати телеметричні дані без використання додаткових комунікаційних модулів. Також мікроконтролер містить до 34 програмованих GPIO, 12-бітний аналого-цифровий перетворювач до 18 каналів, два 8-бітних цифро-аналогових перетворювачі, інтерфейси, модулі PWM для керування виконавчими пристроями, сенсорні входи Touch Sensor, таймери та watchdogs [1-3].

Ще мається вбудована підтримка операційних систем реального часу. Офіційне середовище розробки ESP-IDF використовує як базову RTOS FreeRTOS, яка забезпечує детерміноване виконання задач, підтримку багатозадачності та ефективний розподіл обчислювальних ресурсів між ядрами процесора [1].

Крім того, ESP32 має апаратні засоби криптографічного захисту, зокрема прискорювачі AES, SHA-2, RSA та генератор випадкових чисел RNG, що забезпечує захищену передачу даних і безпечне підключення до мережевої інфраструктури [1].

Розробка ПЗ для ESP32 може виконуватись в середовищах Visual Studio Code, Arduino IDE та PlatformIO. Найбільш функціональним рішенням

вважається ESP-IDF – це офіційний SDK компанії Espressif, який надає засоби роботи з периферією FreeRTOS, мережевими протоколами та апаратними ресурсами мікроконтролера [3].

1.2. Аналіз сучасних підходів до побудови системи автоматичного управління

На сучасному етапі розвитку автоматизації для побудови САУ застосовуються різні підходи, що відрізняються складністю реалізації, вимогами до обчислювальних ресурсів та якістю керування. Найбільш поширеними є ПІД-регулювання, адаптивне керування, нечітке керування та системи з використанням штучного інтелекту.

Найбільш розповсюдженим методом є ПІД-регулювання, що значить пропорційно-інтегрально-диференціальне керування, яке забезпечує високу точність підтримання заданого параметра завдяки поєднанню трьох складових: пропорційної, інтегральної та диференціальної. Основними перевагами цього підходу є простота реалізації, невисокі вимоги до апаратних ресурсів та широка сфера застосування. Разом із тим ефективність ПІД-регулятора значною мірою залежить від правильного налаштування коефіцієнтів, а його робота може погіршуватись за наявності нелінійностей або змін параметрів об’єкта керування [4].

Адаптивне керування використовується у випадках, коли параметри об'єкта можуть змінюватися в процесі роботи. Такі системи здатні автоматично коригувати власні параметри відповідно до поточного стану об'єкта та зовнішніх умов. Перевагою таких систем є здатність забезпечувати стабільну якість керування в умовах невизначеності. Недоліком є складність математичного апарату та підвищені вимоги до обчислювальних ресурсів контролера [5].

Нечітке керування базується на використанні нечіткої логіки та правил виду «if-then». Нечіткі системи можуть працювати з неточними або неповними

даними та враховувати експертні знання при формуванні керуючих впливів. Ефективність нечіткої системи залежить від якості сформованої бази правил [6].

Останніми роками значного поширення набули методи штучного інтелекту, зокрема нейронні мережі. Нейромережеві моделі здатні самостійно виявляти приховані закономірності у вхідних даних та приймати рішення на основі результатів навчання. Розвиток концепцій Edge AI та TinyML дозволив реалізовувати нейромережеві алгоритми навіть на мікроконтролерах з обмеженими ресурсами. Основною перевагою такого підходу є можливість адаптивного реагування на складні та нелінійні процеси. Недоліками залишаються необхідність попереднього навчання моделі та складність інтерпретації результатів її роботи.

1.3. Методи побудови моделей систем автоматичного управління

Побудова модель САУ – один із ключових етапів проєктування сучасних автоматизованих комплексів. Модель дозволяє дослідити поведінку об'єкта керування, оцінити якість регулювання та визначити оптимальні параметри системи ще до її практичної реалізації. Залежно від способу опису об'єкта та принципів формування керуючих впливів використовуються різні методи побудови моделей САУ. Найбільш поширеними є математичне, структурне та імітаційне моделювання.

Математичне моделювання базується на описі об'єкта системою рівнянь, які відображають фізичні процеси та взаємозв'язки між параметрами, що дозволяє отримати точний опис динаміки системи та проводити аналітичні дослідження її стійкості, точності й швидкодії, але такі моделі складні в побудові для нелінійних або багатопараметричних об'єктів. Структурне моделювання передбачає подання системи у вигляді взаємопов'язаних функціональних блоків, де кожен блок виконує окрему функцію, а зв'язки між ними відображають потоки даних та керуючих сигналів. Цей підхід

використовується в основному під час проєктування вбудованих систем, так як дозволяє наочно відобразити архітектуру системи та принципи взаємодії її компонентів. В свою чергу імітаційне моделювання полягає у створенні програмної моделі, яка відтворює роботу реальної системи в умовах, максимально наближених до експлуатаційних, завдяки чому можна досліджувати реакцію системи на різні зовнішні впливи, тестувати алгоритми керування та аналізувати їх ефективність без використання фізичного обладнання.

У сучасних САУ дедалі частіше використовуються інтелектуальні моделі, що поєднують класичні принципи керування з методами штучного інтелекту. Такі моделі здатні враховувати складні залежності між параметрами об'єкта, адаптуватися до зміни умов функціонування та формувати більш ефективні керуючі рішення. Особливо актуальним цей підхід є для вбудованих систем на базі мікроконтролерів, де необхідно забезпечити автономність роботи та мінімізувати обчислювальні витрати.

1.4. Інтеграція елементів штучного інтелекту у вбудовані системи

Сучасний розвиток локальних систем автоматизації можна охарактеризувати переходом від зумовлених алгоритмів до концепції вбудованого ШІ. Такий підхід передбачає, що обчислювальні процедури, пов’язані з аналізом багатовимірних даних, розпізнавання образів і прийняттям рішень, переносяться на периферійні платформи без обов’язкової передачі інформації в хмарні сервіси. Як апаратна платформа для подібних систем найчастіше використовують мікроконтролери, які мають достатню обчислювальну продуктивність та вбудовані засоби комунікації.

Сам же штучний інтелект є комплексом математичних методів і програмних алгоритмів, призначених для реалізації інтелектуальних функцій, які традиційно пов'язують із діяльністю людини: аналізу даних, розпізнавання образів, прогнозування, прийняття рішень і самонавчання. Одним із ключових

принципів ШІ є формування змінних реакцій системи залежно від параметрів зовнішнього середовища.

Для систем автоматичного управління інтеграція елементів ШІ дає низку суттєвих переваг:

- мінімальний час реакції системи;
- автономність функціонування;
- можливість адаптації до змінних умов;
- зниження навантаження на комунікаційні канали;
- підвищення стійкості до відмов мережі;
- реалізація інтелектуального керування в реальному часі.

Реалізація елементів штучного інтелекту на мікроконтролерах супроводжується ресурсними обмеженнями. Порівняно з універсальними процесорами та графічними прискорювачами, мікроконтролери класу SoC (включно з ESP32) мають обмежений обсяг оперативної пам'яті та постійної Flash-пам'яті, тому інтеграція «сильного» ШІ вбудованого класу неможлива. Сучасні інтелектуальні САУ базуються на принципах вузькоспеціалізованого ШІ, який оптимізований під обмежене коло технологічних завдань. Найефективнішим математичним інструментом у такому ШІ є прості нейронні мережі.

Через високу обчислювальну складність алгоритмів математичної оптимізації повноцінне навчання нейронних мереж на самом мікроконтролері є недоцільним. У Edge AI застосовується двоетапна концепція циклу моделі: на першому етапі відбувається формування датасету, проєктування топології мережі, оптимізація функцій активації та розрахунок вагових коефіцієнтів, які виконуються на робочій станції, а на другому етапі отримана математична матриця ваг і зміщень експортується, перетворюється на статичні масиви та компілюється в прошивку мікроконтролера [7].

1.5. Аналіз алгоритмів ШІ

Одним із перспективних напрямів розвитку сучасних вбудованих систем є використання алгоритмів ШІ для автоматизації процесів прийняття рішень, а завдяки розвитку технологій Edge AI та TinyML стало можливим виконання інтелектуальних обчислень безпосередньо на мікроконтролерах з обмеженими апаратними ресурсами.

Із найбільш відомих підходів є використання багатошарових нейронних мереж, зокрема MLP. Цей алгоритм складається з вхідного, одного або декількох прихованих та вихідного шарів нейронів, між якими встановлюються вагові коефіцієнти. Після завершення процесу навчання нейронна мережа здатна виявляти складні нелінійні залежності між вхідними параметрами та формувати відповідні керуючі рішення. Перевагою цього підходу є висока точність класифікації та здатність адаптуватися до складних умов функціонування системи, а також разом із тим для використання нейронної мережі необхідно попередньо виконати її навчання та оптимізацію структури, що потребує додаткових обчислювальних ресурсів на етапі розробки.

Альтернативним підходом є використання Fuzzy Logic. На відміну від нейронних мереж, нечіткі системи приймають рішення на основі набору правил виду «if–then», які формуються експертами предметної області. Такий підхід добре підходить для задач автоматичного керування, де важко встановити чіткі межі між різними режимами роботи системи. Перевагами є простота реалізації, низькі вимоги до обчислювальних ресурсів та висока інтерпретованість результатів, але водночас якість роботи нечіткої системи безпосередньо залежить від повноти та коректності сформованої бази правил [8].

Для задач класифікації та прийняття рішень також можуть використовуватися дерева рішень. Принцип їх роботи полягає у послідовному

виконанні логічних перевірок вхідних параметрів, результатом яких є вибір одного з можливих станів системи. Деревахарактеризуються високою швидкодією, простотою реалізації та незначними вимогами до пам'яті мікроконтролера, а разом із тим їх ефективність знижується при необхідності обробки великої кількості взаємопов'язаних параметрів або складних нелінійних залежностей [8].

Ще одним алгоритмом, який може використовуватися у вбудованих системах, є метод k-Nearest Neighbors. Він базується на пошуку найбільш схожих прикладів у попередньо сформованій навчальній вибірці. Даний метод відносно простий в реалізації та має достатньо високу точність для невеликих наборів даних, проте необхідність зберігання значного обсягу навчальних даних у пам'яті пристрою суттєво обмежує його використання на мікроконтролерах з обмеженими ресурсами [8].

Останніми роками широкого поширення набули технології TinyML, які передбачають використання оптимізованих моделей машинного навчання для виконання безпосередньо на мікроконтролерах. До таких рішень належать квантизовані нейронні мережі, моделі TensorFlow Lite for Microcontrollers та інші спеціалізовані алгоритми. В них є можливість реалізації інтелектуальних функцій за мінімального використання пам'яті та енергоспоживання. Це дозволяє застосовувати технології штучного інтелекту навіть у малопотужних автономних пристроях.

Для задач автоматичного управління процесорами найбільш доцільним є використання багатошарової нейронної мережі. На відміну від нечіткої логіки та дерев рішень, нейромережевий підхід здатний самостійно виявляти приховані закономірності між параметрами системи та адаптуватися до різних режимів функціонування. А також сучасні можливості мікроконтролера ESP32 забезпечують достатню продуктивність для виконання інференсу нейронної мережі малої розмірності в режимі реального часу.

Висновки за розділом 1

Було розглянуто архітектурні особливості та технічні характеристики мікроконтролера ESP32 як платформи для побудови інтелектуальних систем автоматичного управління, яка поєднує достатню обчислювальну продуктивність, розвинену периферію, засоби бездротового зв'язку та підтримку операційних систем реального часу. Все це робить його придатним для реалізації алгоритмів автоматичного керування і локального виконання моделей штучного інтелекту.

Було розглянуто методи керування і принципи інтеграції елементів штучного інтелекту у вбудовані системи і розкрито переваги підходу Edge AI. Також було показано, що через обмежені обчислювальні ресурси мікроконтролерів найбільш доцільним є використання компактних нейромереж.

Також було оцінено та обрано алгоритм ШІ. Порівняльний аналіз поширених алгоритмів (MLP, нечітка логіка, дерева рішень, метод k-найближчих сусідів) показав переваги нейромережевого підходу. Для задачі автоматичного керування процесором найдоцільнішим визначено використання MLP. Вона дозволяє ефективно виявляти приховані нелінійні закономірності між параметрами системи та забезпечує високу точність прийняття рішень у реальному часі за мінімального використання динамічної пам'яті ESP32.

Тому аналіз підтвердив доцільність використання ESP32 як базової платформи для створення інтелектуальної системи автоматичного управління із застосуванням штучних нейронних мереж.

РОЗДІЛ 2
ПРОЄКТУВАННЯ МОДЕЛІ СИСТЕМИ АВТОМАТИЧНОГО УПРАВЛІННЯ ПРОЦЕСОРАМИ

2.1. Концептуальна модель

Розроблювана система призначена для інтелектуального регулювання теплового стану процесорних систем в умовах динамічно змінного навантаження. Основною задачею системи є підтримання допустимого температурного режиму процесора за допомогою адаптивного управління системою охолодження в залежності від температури процесора чи навколишнього середовища, а також поточного обчислювального навантаження. В основі логіки керування лежать нейроні мережі.

Система реалізована на принципах замкненого контуру з від'ємним зворотним зв'язком. Спочатку мікроконтролер виконує циклічне опитування первинних датчиків, реєструючи поточні температури процесора та навколишнього середовища, а також рівень умовного обчислювального навантаження. Отримані метрики піддаються програмній фільтрації для приведення до єдиного інтервалу вхідних ознак нейромережі. Далі нормалізований вектор ознак подається на вхід навченого багатошарового персептрону. На основі математичного розрахунку прямого проходу система виконує нелінійне відображення вхідних параметрів у вектор ймовірностей станів об'єкта. А потім на основі вихідних даних нейромережі з додатковою корекцією за температурою та навантаженням створюються керувальні сигнали для виконавчих механізмів: розраховується кут повороту сервоприводу, а в критичних ситуаціях активується дискретний релейний захист. Паралельно здійснюється візуалізація інформації на OLED-дисплеї.

Для стійкості функціонування САУ серед станів виділено чотири режими роботи.

Таблиця 2.1

Режими роботи САУ

Режим	Характеристика стану	Дії
NORMAL	Низька температура та навантаження	Мінімальне охолодження
WARNING	Підвищення температури або навантаження	Збільшення охолодження
HIGH	Високе навантаження та нагрів	Інтенсивне охолодження
EMERGENCY	Перегрів	Максимальне охолодження

Математична модель системи описує зміну температури процесора, яка формується під динамічним впливом його обчислювального навантаження, температури навколишнього середовища та поточної інтенсивності охолодження.

Математичну модель теплового балансу об'єкта керування, яка визначає зміну температури процесора, можна подати у вигляді диференціального рівняння першого порядку:

$$\tau \frac{dT(t)}{dt} + T(t) = k_1 L(t) + k_2 T_{env}(t) - k_3 C(t), \qquad (2.1)$$

де T(t) – температура процесора, L(t) – обчислювальне навантаження, $T_{env}(t)$ – температура навколишнього середовища, C(t) – інтенсивність охолодження, k_1 k_2 k_3 – коефіцієнти моделі.

Для точного опису нелінійної залежності в САУ інтегровано нейромережу MLP з структурой 3-8-6-4. Вхідний вектор має вид

$$X = [T_{cpu}, T_{env}, L_{cpu}]^T \tag{2.2}$$

Щоб уникнути загасання градієнтів і забезпечити гладкий вихід в прихованих шарах застосовано функцію активації гіперболічного тангенса (tanh):

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.3}$$

Математичний опис функціонування прихованих шарів має вид:

$$H^{(1)} = tanh\left(W^{(1)} * X + B^{(1)}\right), \tag{2.4}$$

$$H^{(2)} = tanh\left(W^{(2)} * H^{(1)} + B^{(2)}\right), \tag{2.5}$$

де $W^{(1)}$, $W^{(2)}$ – матриці вагових коефіцієнтів розмірністю 8x3 та 6x8 відповідно; $B^{(1)}$, $B^{(2)}$ – вектори зміщень шарів.

Вихідний шар мережі містить 4 нейрона, розподіл сигналів на яких формується за допомогою функції Softmax:

$$P_k = \frac{e^{z_k}}{\sum_{i=1}^{4} e^{z_i}}, \qquad z = W^{(3)} * H^{(2)} + B^{(3)}, \tag{2.6}$$

де P_k – розрахункова ймовірність (впевненість мережі) того, чи належить поточний стан до k-го керувального класу ($k \in \{0, 1, 2, 3\}$).

На основі правила вибору максимальної ймовірності визначається індекс домінуючого режиму роботи: $mode = arg \max_k(P_k)$

Рівень охолодження обчислюється на основі базового значення для обраного режиму з додатковою корекцією за температурою та навантаженням. Синтез структури мережі, підбір параметрів і розрахунок вагових матриць W

і B здійснюються на попередньому етапі проєктування на базі сформованого набору даних обсягом 2000 векторів станів.

2.2. Розробка структури та алгоритму датасету

2.2.1. Структура та синтез набору даних

Кожен окремий рядок набору даних є впорядкованим вектором, який містить три вхідних фізичних параметри та одну вихідну цільову ознаку. Вектор вхідних параметрів X формується на основі даних датчиків та органів керування (2.2).

Цільова ознака Y є дискретною величиною та набуває одного з чотирьох значень (від 0 до 3), які відповідають режимам NORMAL, WARNING, HIGH та EMERGENCY.

Синтез набору даних реалізовано за допомогою Python. Фрагмент коду, який виконує генерацію простору ознак та формування міток класів:

```
cpu_temp  = np.random.uniform(20, 90, n)
env_temp  = np.random.uniform(15, 45, n)
load_pct  = np.random.uniform(0, 100, n)
# Цільовий режим
def get_mode(t, l):
   if t < 45 and l < 40:   return 0  # NORMAL
   elif t < 60 and l < 70: return 1  # WARNING
   elif t < 75 and l < 90: return 2  # HIGH
   else:                   return 3  # EMERGENCY
# Цільовий рівень охолодження
def get_cooling(t, l, e):
   base = (t - 20) / 70 * 70 + l * 0.25 + (e - 15) / 30 * 5
   return float(np.clip(base, 0, 100))
modes   = np.array([get_mode(t, l)       for t, l    in zip(cpu_temp, load_pct)])
```

cooling = np.array([get_cooling(t, l, e) for t, l, e in zip(cpu_temp, load_pct, env_temp)])

X = np.column_stack([cpu_temp, env_temp, load_pct])

Для приведення вхідних ознак до єдиного діапазону [0, 1] застосовувалася нормалізація MinMaxScaler з фіксованими межами: мінімальні значення [20, 15, 0], максимальні [90, 45, 100]. Це забезпечує коректну роботу нейромережі як на етапі навчання, так і при інференсі на мікроконтролері. Далі за допомогою MLPClassifier (активація tanh, два прихованих шари по 8 та 6 нейронів) було навчено модель класифікації. Після навчання ваги та зміщення було експортовано в масиви.

2.2.2. Алгоритм нормалізації даних

Як функції активації прихованих шарів використовується гіперболічний тангенс. Його чутлива область знаходиться поблизу нуля. Якщо подати в мережу сирі фізичні значення, то нейрон швидко переходить у режим насичення, градієнти згасають, і точність передбачень падає. Для вирішення цієї проблеми було застосовано алгоритм MinMax-масштабування, який перетворює вихідні фізичні діапазони на безрозмірний інтервал [0; 1]. Математична формула нормалізації для кожної ознаки має вигляд:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}, \tag{2.7}$$

Межі нормалізації зафіксовано на етапі підготовки набору даних і жорстко задано як у навчальному скрипті на Python, так і в коді ESP32. В іншому разі масштаби ознак на мікроконтролері будуть спотворені, що порушить роботу системи.

Зафіксовані граничні значення ознак:

T_{cpu}: [20; 90] ℃ (DS18B20)

T_{env}: [15; 45] ℃ (DHT22)

L_{cpu}: [0; 100] % (потенціометр)

2.3. Програмна реалізація

Програмне забезпечення ESP32 розподілено між двома файлами: головного циклу керування та модуль нейромережі.

2.3.1. Головний цикл управління

Головний цикл реалізований у функції loop() з фіксованим тактом 1000 мс, що забезпечується через порівняння поточного значення системного таймера millis() з попередньою міткою часу. На кожній ітерації послідовно виконуються п'ять етапів: збір даних, їх попередня обробка, нейромережевий інференс, керування виконавчими пристроями та виведення інформації.

Перший етап – це збір даних з датчиків. Датчик DS18B20 передає температуру процесорного вузла через інтерфейс 1-Wire за допомогою бібліотек OneWire та DallasTemperature. Датчик DHT22 надає температуру навколишнього середовища через однопровідний цифровий протокол. Сам рівень умовного навантаження формується потенціометром, а значення 12-бітного АЦП перетворюється на відсотки за формулою:

$$L_{cpu} = \frac{ADC}{4095} * 100\%, \tag{2.8}$$

де ADC – поточне значення аналого-цифрового перетворювача, 4095 – максимальне значення для 12-бітного АЦП ESP32.

Для відтворення теплової поведінки процесора застосовується модель динамічного нагрівання і базується на тому, що температура процесора зростає пропорційно навантаженню:

$$T_{cpu} = T_{DS18B20} + \frac{L_{cpu}}{100} * 65, \tag{2.9}$$

де $T_{DS18B20}$ – базова температура, виміряна датчиком, 65 – максимальний перепад температури при повному навантаженні.

На другому етапі перед подачею даних у нейромережу вхідні значення проходять обробку і фільтрацію. У разі виявлення збоїв система автоматично підставляє значення за замовчуванням 35 для температури процесора та 24 для

температури середовища. Для усунення короткочасних шумів і стрибків застосовується алгоритм експоненційного ковзного середнього:

$$T_{filt}(t) = \alpha * T_{filt}(t-1) + (1-\alpha) * T_{cur}(t), \quad (2.10)$$

де $\alpha = 0{,}7$ – коефіцієнт згладжування, $T_{filt}(t-1)$ – попереднє відфільтроване значення, $T_{cur}(t)$ – поточне вимірювання. Коефіцієнт $\alpha = 0{,}7$ означає, що 70% ваги має накопичена історія вимірювань і лише 30% припадає на нове значення, що задезпечує ефективне придушення імпульсних завад при збереженні достатньої швидкості реакції системи.

На третьому етапі відфільтровані та нормалізовані дані передаються у функцію predict_system_control(), яка реалізує прямий прохід нейронної мережі і повертає структуру AI_Output.

Четвертий етап – на основі результату інференсу формується три управляючі впливи. Кут повороту сервоприводу розраховується лінійним перетворенням рівня охолодження в кутовий діапазон сервоприводу:

$$\theta = \frac{C_{level}}{100} * 180^{\circ}, \quad (2.11)$$

де C_{level} – рівень охолодження.

Реле керується безпосередньо нейромережею з використанням алгоритму гістерезису. Логіка увімкнення та вимкнення реле є асиметричною:

$$Relay_{ON} = (mode = 3) \wedge (P_{mode} > 0.9) \wedge \left(x_{cpu} > 0.7\right), \quad (2.12)$$

$$Relay_{OFF} = (mode < 2) \wedge (mode = 2 \wedge P_{mode} < 0.75), \quad (2.13)$$

де x_{cpu} – нормалізоване значення температури процесора, P_{mode} – впевненість нейромережі у поточному класі. Реле вмикається лише при одночасному виконанні трьох умов: класифікація режиму як EMERGENCY, впевненість мережі понад 90% та нормалізована температура процесора понад 0,70 (що відповідає $T_{cpu} > 69°C$). Вимикається реле лише після впевненого переходу системи у режим нижче HIGH або при зниженні впевненості в

режимі HIGH нижче 75%. Така асиметрія порогів забезпечує стійку поведінку реле без хибних спрацювань на межах між режимами.

Тобто воно вмикається або коли нейромережа визначає максимальний рівень охолодження, або коли температура процесора незалежно перевищує критичну межу 85°C, що реалізує апаратний захист від перегріву незалежно від стану нейромережі.

І на п’ятому етапі – виведення інформації.

2.3.2. Модуль нейромережі

Нейромережевий модуль реалізований у файлі ai_model.h. Він містить статичні масиви вагових коефіцієнтів, функції активації та функцію прямого проходу predict_system_control().

Перед подачею на вхідний шар значення ознак перетворюються до безрозмірного діапазону [0; 1] за формулою (2.7).

Граничні значення зафіксовані. Нормалізований вхідний вектор на (2.2).

Перший прихований шар містить 8 нейронів. Для кожного нейрона і обчислюється зважена сума вхідного вектора та застосовується функція активації гіперболічного тангенса:

$$z_i^{(1)} = \sum_{j=0}^{2} W_{ij}^{(1)} * x_j + B_i^{(1)}, \; i = 0 \dots 7, \tag{2.14}$$

$$h_i^{(1)} = tanh\left(z_i^{(1)}\right), \tag{2.15}$$

де $W^{(1)}$ – матриця вагових коефіцієнтів розміром 8x3, $B^{(1)}$ – вектор зміщень розміром 8x1.

Другий прихований шар містить 6 нейронів і приймає на вхід вектор активацій першого шару $H^{(1)}$:

$$z_i^{(2)} = \sum_{j=0}^{7} W_{ij}^{(2)} * H_i^{(1)} + B_i^{(2)}, \; i = 0 \dots 5, \tag{2.16}$$

де $W^{(2)}$ – матриця вагових коефіцієнтів розміром 6x8, $B^{(2)}$ – вектор зміщень розміром 6x1.

Вихідний шар містить 4 нейрони – по одному на кожний клас стану системи. Для кожного класу k обчислюється логіт – значення лінійної комбінації без функції активації:

$$z_k^{(3)} = \sum_{j=0}^{5} W_{kj}^{(3)} * H_j^{(2)} + B_k^{(3)}, \; k = 0...3, \tag{2.17}$$

де $W^{(3)}$ – матриця вагових коефіцієнтів розміром 4x6, $B^{(3)}$ – вектор зміщень розміром 4x1.

Для перетворення логітів на вектор імовірностей застосовується функція Softmax. Для чисельної стабільності застосовується модифікований варіант із відніманням максимального логіта:

$$P_k = \frac{e^{z_k^{(3)} - z_{max}}}{\sum_{m=0}^{3} e^{z_m^{(3)} - z_{max}}}, z_{max} = \max_m \left(z_m^{(3)}\right), \tag{2.18}$$

Клас поточного режиму роботи визначається правилом вибору максимальної ймовірності:

$$mode = arg\max_k(P_k), \tag{2.19}$$

Значення P_{mode} інтерпретується як коефіцієнт впевненості мережі у прийнятому рішенні та виводиться на OLED-дисплей як діагностичний параметр.

Рівень охолодження C_{level} розраховується на основі базового значення для визначеного режиму з адаптивною поправкою, яка залежить від нормалізованих значень температури та навантаження:

$$C_{level} = C_{base} + \Delta C * P_{mode}, \tag{2.20}$$

$$\Delta C = x_{cpu} * 15 + x_{load} * 10 + x_{env} * 5, \tag{2.21}$$

де C_{base} – масив базових рівнів охолодження для кожного режиму; ΔC – забезпечує плавний перехід між режимами; множник P_{mode} масштабує

поправку пропорційно впевненості мережі – чим вища впевненість, тим сильніший вплив поправки.

Загальна обчислювальна складність одного інференсу становить $O(N_1 * N_{in} + N_2 * N_1 + N_{out} * N_2)$, $N_{in} = 3$, $N_1 = 8$, $N_2 = 6$, $N_{out} = 4$, що дорівнює 98 операціям множення та додавання з рухомою комою.

Висновки за розділом 2

У розділі було розроблено концептуальну модель системи автоматичного управління процесорами на ESP32 з використанням нейронної мережі. Ще було сформовано набір даних для роботи нейромережі, підготовлено вхідні параметри та розроблено модель нейромережі для прийняття керуючих рішень. Також було виконано програмну реалізацію основних модулів системи та підтверджено можливість виконання нейромережевого інференсу безпосередньо на мікроконтролері.

РОЗДІЛ 3
МОДЕЛЮВАННЯ, ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1. Інструментарій розробки програмного забезпечення та середовище симуляції

Для розробки інтелектуальної системи було обрано комплекс програм, які забезпечують ефективне проєктування, налагодження, моделювання та тестування вбудованої системи керування на ESP32. При доборі інструментів ключовими критеріями стали повноцінна підтримка архітектури ESP32, можливість детального симулювання периферійних пристроїв, а також зручність налагодження вбудованого ПЗ, швидкість прототипування та загальна доступність засобів розробки. Як основне середовище для написання коду було обрано платформу Arduino IDE. Код реалізовано на C++.

Для моделювання апаратної частини проєкту було обрано онлайн-платформу Wokwi. Усередині середовища було зібрано віртуальну інтелектуальну систему охолодження, яка об'єднала в єдину схему мікроконтролер ESP32, датчик температури DHT22, цифровий термодатчик DS18B20, OLED-дисплей SSD1306, сервопривід, потенціометр для імітації обчислювального навантаження та релейний модуль.

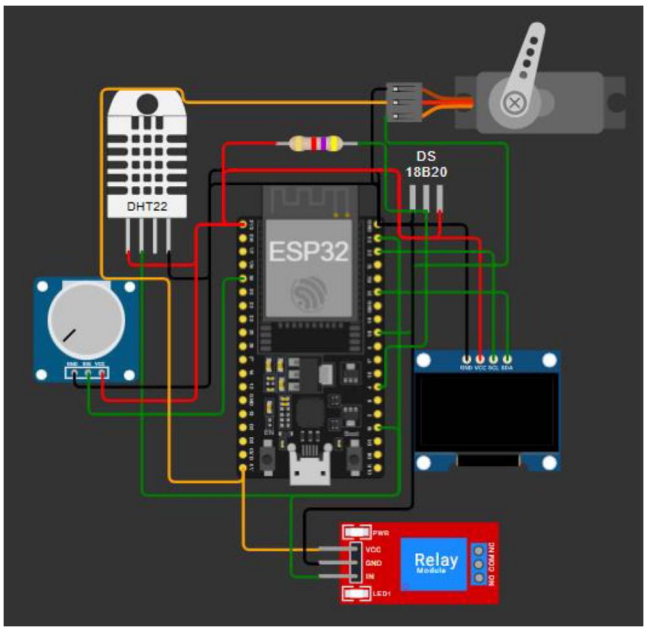


Рисунок 3.1 – Схема інтелектуальної системи.

Керування OLED-дисплеєм SSD1306 було організовано на основі зв'язки бібліотек Adafruit_GFX та Adafruit_SSD1306, які забезпечують виведення текстової та графічної інформації через послідовний інтерфейс I2C. Взаємодія з температурними датчиками в режимі реального часу потребувала диференційованого підходу: для роботи з DHT22 застосовувалася стандартна DHT sensor library, а для отримання точних даних з цифрового термометра DS18B20 використовувався дует бібліотек OneWire та DallasTemperature, які підтримують протокол 1-Wire. Фізичне керування виконавчими механізмами виконувалося за допомогою бібліотеки ESP32Servo. Сама нейромережева модель була повністю написана на C++ та винесена в окремий файл ai_model.h. Вся математика мережі була реалізована на рівні базових арифметичних операцій, без використання фреймворків. Для наскрізного аналізу та налагодження розробленого комплексу активно застосовувалися вбудовані інструменти Arduino IDE – Serial Monitor та Serial Plotter. Дані безперервно транслювалися через послідовний порт у структурованому форматі.

3.2 Результати моделювання

Для перевірки розробленої інтелектуальної САУ було проведено симуляцію роботи системи у середовищі Wokwi. Сценарій моделювання

охоплює послідовне зростання навантаження від мінімального до критичного рівня, що дозволяє оцінити поведінку системи в усьому робочому діапазоні. Частота циклу керування становить 1000 мс. Температуру навколишнього середовища T_{env} для зручності була стабільно на рівні 30°C.

Таблиця 3.1

Вихідні дані

t, мс	T_{cpu}, °C	L_{cpu}, %	C_{level}, %	Режим ШІ	Очікуваний режим ШІ	P_{mode}	θ, °	Реле
3 000	29.2	4.4	14.9	0	0	1	26	0
4 000	29.2	13.3	15.8	0	0	1	28	0
5 000	36.5	30.4	19.0	0	0	1	34	0
6 000	38.8	34.1	19.7	0	0	0.98	35	0
7 000	41.8	41.4	49.4	1	1	0.83	88	0
8 000	55.4	60.0	56.0	1	1	0.99	100	0
9 000	60.3	67.7	86.4	2	1	0.91	155	0
10 000	68.2	80.7	90.8	2	2	1	163	0
11 000	71.1	85.9	86.7	2	2	0.76	155	0
12 000	73.8	89.3	100.0	3	2	0.91	180	1
13 000	78.4	96.7	100.0	3	3	1	180	1

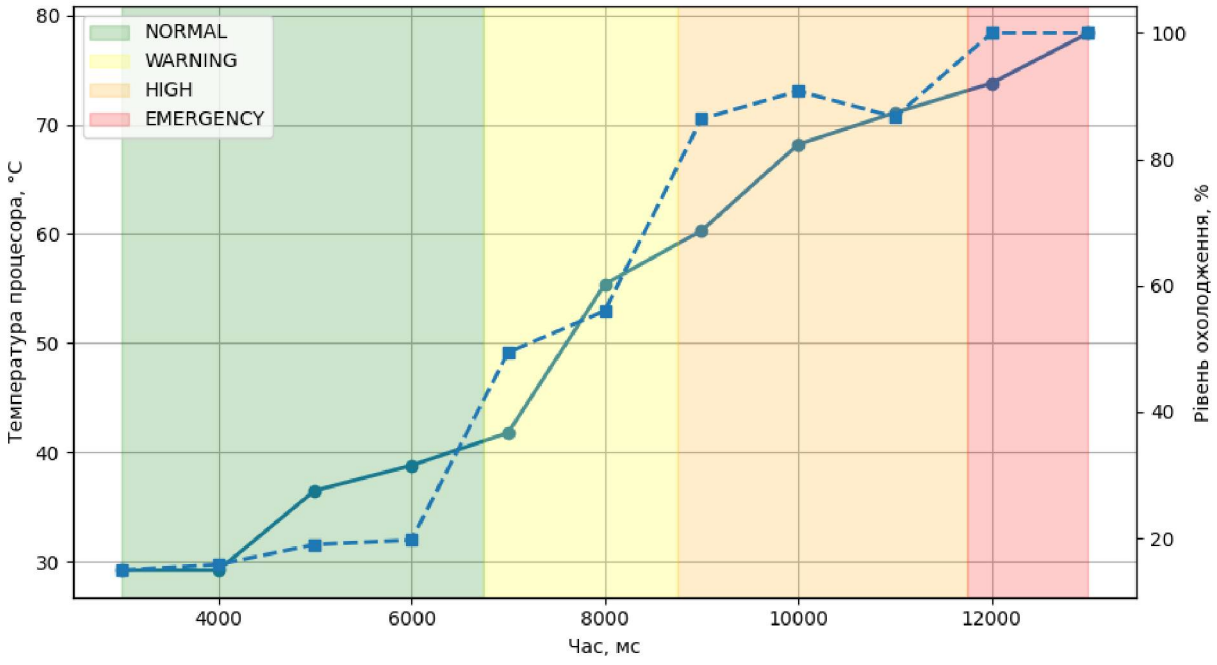


Рисунок 3.2 – Залежність рівня охолодження від температури.

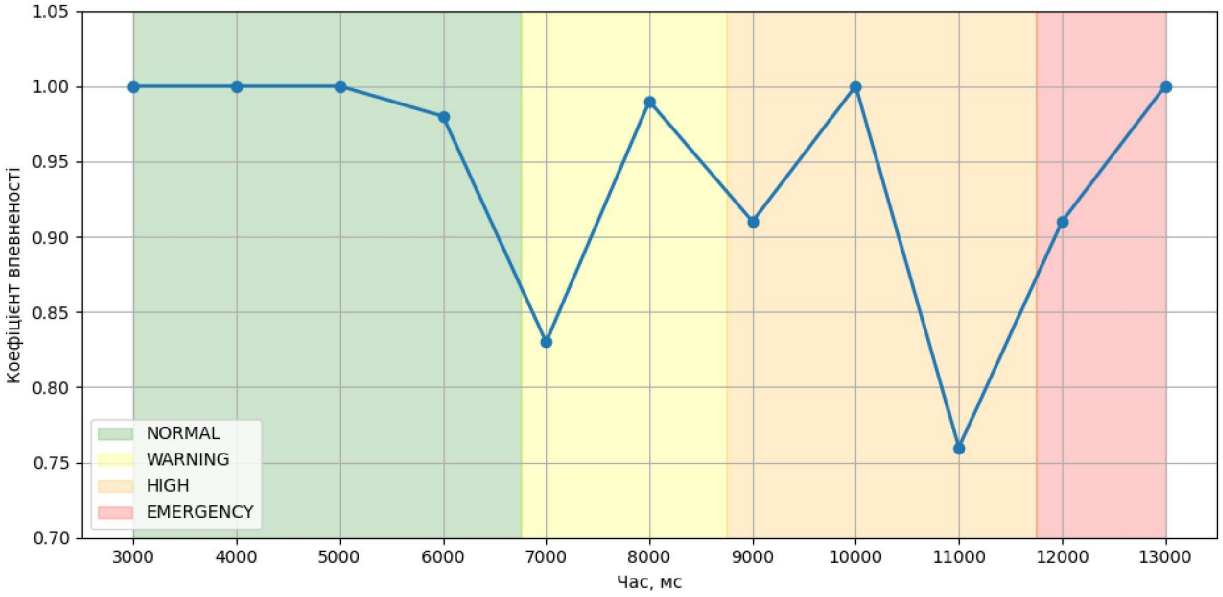


Рисунок 3.3 – Динаміка впевненості нейромережі.

Аналіз отриманих даних дозволяє виділити кілька характерних аспектів роботи системи.

Режим NORMAL (3 751 - 6 751 мс): при низькому навантаженні L_{cpu} = 4,4…34,1% та температурі процесора T_{cpu} = 29,2…38,8°C нейромережа впевнено класифікує стан як NORMAL з коефіцієнтом впевненості P_{mode} = 0,98…1. Рівень охолодження підтримується в діапазоні 14,9-19,7%, кут сервоприводу 26-35°. Реле аварійного захисту відключене. Класифікація нейромережі повністю збігається з очікуваним режимом.

Перехід до режиму WARNING (7 751 - 8 751 мс): при зростанні навантаження до L_{cpu} = 41,4% та температурі T_{cpu} = 41,8°C фіксується перший перехід між режимами. Коефіцієнт впевненості у момент переходу знижується до 0,83, що є закономірною поведінкою класифікатора на основі Softmax: вхідний вектор знаходиться у перехідній зоні між класами NORMAL та WARNING. Рівень охолодження стрибкоподібно зростає до 49,4%, кут сервоприводу до 88°, що демонструє превентивну реакцію системи на зростаюче навантаження. При подальшому зростанні навантаження до 60% впевненість відновлюється до 0,99.

Режим HIGH (9 751 мс): на дев'ятому такті нейромережа класифікує стан як HIGH при T_{cpu} = 60,3 °C та L_{cpu} = 67,7%, тоді як детермінована еталонна логіка ще визначає цей стан як WARNING. Це свідчить про те, що нейромережа навчилась враховувати комбінований вплив температури та навантаження і реагує на загрозу перегріву з випередженням. Рівень охолодження сягає 86,4%, кут сервоприводу – 155°.

Режим EMERGENCY: (12 751 – 13 751 мс): при T_{cpu} = 73,8 °C та L_{cpu} = 89,3% нейромережа переходить до режиму EMERGENCY раніше за еталонну логіку, яка ще визначає стан як HIGH. Рівень охолодження досягає максимального значення 100,0%, кут сервоприводу – 180°, реле активується нейромережею відповідно алгоритму гістерезису: одночасне виконання умов режиму EMERGENCY, впевненості мережі 0,91 > 0,9 та нормалізованої температури $x_{cpu} > 0,7$ забезпечує надійне спрацювання без хибних активацій. При подальшому зростанні навантаження до 96,7% та температури до 78,4 °C еталонний і нейромережевий режими збігаються – обидва визначають стан як EMERGENCY з впевненістю 1,00.

З 11 вимірювань у 9 випадках класифікація нейромережі повністю збіглась з еталонною логікою. У 2 випадках нейромережа перейшла до більш серйозного режиму раніше за еталон: система завчасно збільшує інтенсивність охолодження, запобігаючи досягненню критичних температурних порогів. Середня впевненість мережі за весь сеанс склала 0,95, мінімальне значення 0,76 зафіксовано лише в одному перехідному вимірюванні.

Висновки за розділом 3

В цьому розділі проведено моделювання та дослідження роботи розробленої системи автоматичного управління у середовищі Wokwi із використанням програмного забезпечення, реалізованого на платформі ESP32. Для перевірки працездатності системи була зроблена зміна навантаження від мінімальних до критичних значень та проаналізовано реакцію

нейромережевого алгоритму керування. Результати показали, що система визначає режими роботи та забезпечує адаптивне регулювання інтенсивності охолодження залежно від поточного стану. У більшості випадків рішення нейронної мережі повністю збігалися з еталонною логікою. Середня впевненість класифікації нейронної мережі склала близько 0,95.

Отже, результати моделювання підтвердили працездатність і ефективність розробленої моделі інтелектуальної системи автоматичного управління та довели доцільність використання нейромережевого підходу для інтелектуального управління на платформі ESP32.

ВИСНОВКИ

В даній роботі розроблено та досліджено модель системи автоматичного управління процесорами на базі мікроконтролера ESP32 із застосуванням елементів штучного інтелекту. Був проведений аналіз архітектури ESP32, що підтвердив придатність цієї платформи для реалізації вбудованого штучного інтелекту. Було реалізовано двоетапну концепцію Edge AI: нейронна мережа навчається на робочій станції, після чого вагові коефіцієнти експортуються у статичні масиви та компілюються мікроконтролер у вигляді автономного інференс-модуля. Розроблено нейромережевий класифікатор на основі багатошарового персептрона. Також ще було реалізовано замкнений контур управління .

Результати комп'ютерного моделювання у середовищі Wokwi продемонстрували роботу системи: точність класифікації режимів склала 9 з 11 вимірювань, середня впевненість нейромережі – 0,95. Ця модель демонструє практичну доцільність застосування нейромережевих методів у вбудованих системах автоматичного управління.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Espressif Systems. ESP32 Series Datasheet (v5.2) [Електронний ресурс]. – Режим доступу: URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

2. Espressif Systems. ESP32 Arduino Core's Documentation [Електронний ресурс]. – Режим доступу:

URL: https://docs.espressif.com/projects/arduino-esp32/en/latest/

3. Oner V.O. Developing IoT Projects with ESP32. – Packt Publishing, 2023. – 398 с.

4. Іванов А.О. Теорія автоматичного керування. Дніпропетровськ: Національний гірничий університет. – 2003. – 250 с.

5. Huzmezan M. Adaptive control for helicopter vibration reduction using acceleration feedback / M. Huzmezan, J. Maciejowski // Proceedings of the American Control Conference. – 2002. – P. 1235–1240.

6. Tanaka K. Fuzzy Control Systems Design and Analysis: A Linear Matrix Inequality Approach / K. Tanaka, H. O. Wang. – New York : John Wiley & Sons, 2004. – 320 p.

7. Rovai M. Edge AI Engineering: A Practical Guide for Embedded Machine Learning Applications / M. Rovai. – 2023.

8. Han J., Kamber M., Pei J. Data Mining: Concepts and Techniques. 4th ed. Morgan Kaufmann, 2022.

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних систем та робототехніки
Максим ХРУСЛОВ
_____
(підпис)
«13» жовтня 2025року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувача першого (бакалаврського) рівня вищої освіти
(першого (бакалаврського) / другого (магістерського))

НИКОЛАЄНКА ОЛЕКСІЯ ЮРІЙОВИЧА
(прізвище, ім'я, по батькові студента)

Спеціальність: 151 – Автоматизація, комп'ютерно-інтегровані технології
Освітня програма: «Автоматизація, комп'ютерно-інтегровані технології»

1. Тема роботи **МОДЕЛЬ СИСТЕМИ АВТОМАТИЧНОГО УПРАВЛІННЯ ПРОЦЕСОРАМИ НА БАЗІ ESP32 З ЗАСТОСУВАННЯМ ЕЛЕМЕНТІВ ШТУЧНОГО ІНТЕЛЕКТУ**
керівник роботи **Рало Олександр Миколайович, старший викладач зво**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***№ 4101-5/1245 від 28.05.2026 року.***
2. Строк подання студентом роботи ***30 травня 2026 року***
3. Вихідні дані до роботи:
   1) архітектурна специфікація та технічні характеристики мікроконтролера ESP32;
   2) середовище симуляції для моделювання вбудованих систем;
   3) інструментарій розробки програмного забезпечення;
   4) датчики фізичних параметрів середовища;
   5) виконавчі пристрої для систем автоматичного управління;
   6) базові принципи побудови та навчання нейронних мереж малої розмірності.
4. Перелік питань, які потрібно розробити:
   1) Аналіз сучасних підходів до побудови систем автоматичного управління на базі мікроконтролерів та дослідження архітектурних особливостей платформи ESP32 для задач керування.
   2) Дослідження методів побудови моделей автоматичного управління та визначення можливостей інтеграції елементів штучного інтелекту у вбудовані системи.
   3) Огляд та аналіз алгоритмів штучного інтелекту, придатних для реалізації на мікроконтролерах з обмеженими ресурсами (нейронні мережі малої розмірності, нечітка логіка, алгоритми прийняття рішень).
   4) Розробка концептуальної моделі системи автоматичного управління процесорами на базі ESP32 з використанням елементів штучного інтелекту.
   5) Розробка структурної схеми системи, опис функціональних модулів та принципів їх взаємодії.
   6) Проведення моделювання роботи системи та аналіз отриманих результатів.
   7) Формування рекомендацій щодо практичного впровадження розробленої моделі у системах автоматизації.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми роботи та узгодження індивідуального завдання	01.10.2025 – 10.10.2025
2	Літературний огляд з проблематики систем автоматичного управління та архітектури ESP32	11.10.2025 – 15.11.2025
3	Аналіз методів штучного інтелекту для вбудованих систем та вибір підходів для реалізації моделі	16.11.2025 – 20.12.2025
4	Розробка концептуальної моделі системи автоматичного управління	21.12.2025 – 31.01.2026
5	Розробка структурної схеми системи та опис алгоритмів функціонування	01.02.2026 – 15.03.2026
6	Моделювання роботи системи та аналіз результатів	16.03.2026 – 15.04.2026
7	Підготовка рекомендацій щодо застосування моделі	16.04.2026 – 30.04.2026
8	Підготовка і оформлення звітних матеріалів та додатків кваліфікаційної роботи. Оформлення списку літератури	01.05.2026 – 20.05.2026
9	Оформлення пояснювальної записки відповідно до встановлених вимог	01.05.2026 – 30.05.2026
10	Предзахист кваліфікаційної роботи	10.05.2026 – 20.05.2026
11	Остаточне оформлення звіту	до 30.05.2026
12	Представлення кваліфікаційної роботи керівнику	30.05.2026

5. Дата видачі завдання ***13 жовтня 2025 року.***

Здобувач вищої освіти _____ Олексій НИКОЛАЄНКО
(підпис) (ім'я, прізвище)

Керівник роботи _____ Олександр РАЛО
(підпис) (ім'я, прізвище)

Додаток Б

Затверджую

_____

«_____»_____2026 р.

Технічне завдання

на розробку програмного виробу «МОДЕЛЬ СИСТЕМИ АВТОМАТИЧНОГО УПРАВЛІННЯ ПРОЦЕСОРАМИ НА БАЗІ ESP32 З ЗАСТОСУВАННЯМ ЕЛЕМЕНТІВ ШТУЧНОГО ІНТЕЛЕКТУ»

1.	Введення	1.1. Назва: Модель системи автоматичного управління процесорами на базі ESP32 з застосуванням елементів штучного інтелекту. 1.2. Галузь застосування: комп'ютерні технології.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101-5/225 від «30» травня 2026 р. (Додаток А до пояснюва-льної записки).
3.	Призначення розробки	3.1. Мета розробки: моделювання інтелектуальної системи автоматичного управління процесами з використанням штучного інтелекту. 3.2. Розробка призначена для автоматичного аналізу параметрів функціонування та формування керуючих впливів на систему з використанням елементів штучного інтелекту, система забезпечує моніторінг параметрів, оцінку поточного режиму роботи та автоматичне керування виконавчими пристроями. 3.3. Вихідні дані розробки: архітектурна специфікація та технічні характеристики ESP32; середовище симуляції; інструментарій розробки; датчики фізичних параметрів середовища; виконавчі пристрої для САУ; базові принципи побудови та навчання нейронних мереж.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: програмний виріб повинен забезпечувати реалізацію нейромережевої моделі прийняття рішень для системи автоматичного управління, обробку вхідних параметрів та визначення режиму функціонування системи відповідно до результатів роботи нейромережі. 4.2. Вимоги до надійності: забезепчувати стабільну роботу системи автоматичного управління, коректне функціонування нейромережевої моделі та своєчасне реагування на зміну вхідних параметрів.

		4.3.Вимоги до умов експлуатації: передбачається на базі ESP32 при стандартних умовах функціонування електронного обладнання. 4.4. Вимоги до складу і параметрів технічних засобів: звичайне обчислювальне обладнання, ПК, тощо 4.5. Вимоги до інформаційної та програмної сумісності: сумісність із середовищем Arduino IDE та платформою ESP32. 4.6. Вимоги до маркування та упаковки: немає 4.7. Вимоги до транспортування і зберігання: на звичайних носіях інформації 4.8. Спеціальні вимоги: немає.
5.	Вимоги до програмної документації	Програмною документацією до виробу «МОДЕЛЬ СИСТЕМИ АВТОМАТИЧНОГО УПРАВЛІННЯ ПРОЦЕСОРАМИ НА БАЗІ ESP32 З ЗАСТОСУВАННЯМ ЕЛЕМЕНТІВ ШТУЧНОГО ІНТЕЛЕКТУ» вважати: 1) Справжнє Технічне завдання на розробку виробу (Додатку Б до пояснювальної записки). 2) Опис математичної моделі системи автоматичного управління та нейромережевого алгоритму (розділ 2 пояснювальної записки) 3) Опис реалізації системи та результатів моделювання (розділ 3 пояснювальної записки)
6.	Вимоги до техніко-економічних показників	Спеціальні вимоги до техніко-економічних показників не висувалися.
7.	Стадії і етапи розробки	01.10.2025-10.10.2025 – Затвердження теми роботи та узгодження індивідуального завдання 11.10.2025-15.11.2025 – Літературний огляд з проблематики систем автоматичного управління та архітектури ESP32 16.11.2025-20.12.2025 – Аналіз методів штучного інтелекту для вбудованих систем та вибір підходу до реалізації моделі 21.12.2025-31.01.2026 – Розробка концептуальної моделі системи автоматичного управління 01.02.2026-15.03.2026 – Розробка структурної схеми системи та опис алгоритмів функціонування 16.03.2026-15.04.2026 – Моделювання роботи системи та аналіз отриманих результатів 16.04.2026-30.04.2026 – Формування рекомендацій щодо практичного застосування розробленої моделі

		01.05.2026-20.05.2026 – Підготовка та оформлення пояснювальної записки, додатків і списку використаних джерел 10.05.2026-20.05.2026 – Попередній захист кваліфікаційної роботи до 30.05.2026 – Остаточне оформлення матеріалів кваліфікаційної роботи 30.05.2026 – Представлення кваліфікаційної роботи керівнику
8.	Порядок контролю і приймання програмного продукту	1. Контроль виконання згідно календарного плану; 2. Захист на засіданні Екзаменаційної комісії; 3. Подання пояснювальної записки у паперовому та електронному вигляді.

Виконавець
студент групи КУ-41
НИКОЛАЄНКО О.Ю.
_____

Замовник
старший викладач зво
РАЛО О.М.
_____

Додаток В

Програма і методика випробувань програмного виробу

«Модель системи автоматичного управління процесорами на базі ESP32 з застосуванням елементів штучного інтелекту»

1. Об'єкт випробувань

1. Назва програмного виробу: «Модель системи автоматичного управління процесорами на базі ESP32 з застосуванням елементів штучного інтелекту»

2. Галузь застосування: Автоматизація та комп'ютерно-інтегровані технології

3. Перераховані відомості запозичуються з відповідних розділів.

2. Мета випробувань

Перевірка відповідності функціональності розробленої моделі інтелектуальної системи автоматичного управління заявленим вимогам: коректність роботи нейромережевого класифікатора, правильність формування управляючих впливів на виконавчі пристрої та адекватність реакції системи на зміну вхідних параметрів у всіх режимах функціонування.

3. Загальні положення

1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

2. Місце і тривалість випробувань

Приймальні випробування проводяться на базі комп'ютерного класу кафедри комп'ютерних систем та робототехніки в період роботи атестаційної комісії.

3. Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі, що відповідає цій програмі і методиці випробувань.

4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програмного виробу

Модель системи автоматичного управління повинна задовольняти наступним вимогам:

1. виконуватись у середовищі симуляції Wokwi без необхідності фізичного апаратного забезпечення;
2. коректно зчитувати показники датчиків і класифікувати режими;
3. формувати управляючий сигнал для сервоприводу в діапазоні 0°–180°;
4. коректно активувати реле на основі рішення нейромережі;
5. відображати поточний режим роботи, температуру, рівень охолодження та впевненість нейромережі на дисплеї;
6. забезпечувати стійку роботу при некоректних показниках датчиків шляхом підстановки значень за замовчуванням;
7. вимоги до маркування та упаковки не висуваються;
8. вимоги до транспортування і зберігання не висуваються.

5. Вимоги до програмної документації

Програмною документацією до виробу «Модель системи автоматичного управління процесорами на базі ESP32 з застосуванням елементів штучного інтелекту» вважати:

1. пояснювальну записку до кваліфікаційної роботи з описом концептуальної моделі, математичного апарату та програмної реалізації;
2. програму і методику випробувань розробленої системи (представити як Додаток А до пояснювальної записки);

3. текст програми – файли sketch.ino та ai_model.h (Додаток Б і Додаток В відповідно).

6. Засоби і порядок випробувань

6.1. Засоби випробувань

Для проведення випробувань необхідні:

- комп'ютер з доступом до мережі Інтернет та браузером (Chrome, Firefox тощо);
- онлайн-середовище симуляції Wokwi (https://wokwi.com) з завантаженим проєктом;
- файли програмного виробу: sketch.ino, ai_model.h, diagram.json.

6.2. Порядок проведення випробувань

Випробування проводяться у два етапи.

1-й етап – ознайомчий:

1. Перевірка комплектності програмної документації на відповідність вимогам.
2. Перевірка комплектності складу програмних файлів: наявність sketch.ino, ai_model.h, diagram.json.
3. Перевірка коректності завантаження проєкту у середовищі Wokwi та відображення віртуальної схеми.

2-й етап — випробування програмного виробу:

1. Перевірка відповідності технічних характеристик програми вимогам.
2. Перевірка ступеня виконання функціональних вимог до програми.
3. Порядок запуску: відкрити проєкт у середовищі Wokwi та натиснути кнопку «Start Simulation». Після запуску симуляції у вікні Serial Monitor відображаються дані, на OLED-дисплеї – поточний стан системи.
4. Зміна вхідного параметра навантаження здійснюється обертанням потенціометра у віртуальній схемі.

7. Перелік тестів

Тест 1 – Режим NORMAL

Встановити потенціометр у положення, що відповідає навантаженню Lcpu<40%.

1. Дочекатися стабілізації показників (2–3 такти).
2. Перевірити у Serial Monitor: поле AI_Mode = 0, State_Name = NORMAL, Relay_State = 0.
3. Перевірити на OLED-дисплеї: відображення режиму NORMAL, рівень охолодження у діапазоні 10-25%.

Тест вважається пройденим, якщо система класифікує стан як NORMAL, реле відключене, сервопривід знаходиться у діапазоні 0°-45°.

Тест 2 – Режим WARNING / HIGH

1. Встановити потенціометр у положення, що відповідає навантаженню Lcpu=50-75%.
2. Дочекатися стабілізації показників (2–3 такти).
3. Перевірити у Serial Monitor: поле AI_Mode = 1 або AI_Mode = 2, Relay_State = 0.
4. Перевірити рівень охолодження: Clevel=40-75%, кут сервоприводу 70°-135°.

Тест вважається пройденим, якщо система коректно підвищує інтенсивність охолодження при зростанні навантаження, реле залишається відключеним.

Тест 3 – Режим EMERGENCY

1. Встановити потенціометр у максимальне положення (Lcpu>90 %).
2. Дочекатися стабілізації показників (2-3 такти).
3. Перевірити у Serial Monitor: поле AI_Mode = 3, State_Name = EMERGENCY, Confidence > 0.9, Relay_State = 1.

4. Перевірити рівень охолодження: Clevel=100%, кут сервоприводу 180°.

5. Після зниження навантаження до Lcpu = 50-70% перевірити що реле залишається увімкненим протягом 2-3 тактів.

Тест вважається пройденим, якщо система переходить до режиму EMERGENCY, рівень охолодження досягає 100%, реле активується при впевненості > 90%, та не вимикається миттєво при зниженні навантаження.

Тест 4 – Захист від некоректних даних датчиків

1. У файлі sketch.ino тимчасово встановити примусове значення cpuTemp = NaN для імітації відмови датчика.

2. Запустити симуляцію та перевірити у Serial Monitor, що система підставляє значення за замовчуванням 35,0 °C і продовжує роботу без зупинки.

3. Відновити оригінальний код.

Тест вважається пройденим, якщо система не зупиняється при відмові датчика та коректно підставляє резервне значення.

Висновки: Тест 1 успішно пройшов випробування. Тест 2 успішно пройшов випробування. Тест 3 успішно пройшов випробування. Тест 4 успішно пройшов випробування. Випробування пройшло успішно, розроблена модель інтелектуальної системи автоматичного управління відповідає заявленим функціональним вимогам.

Виконавець: студент групи КУ-41, Николаєнко О.Ю. _____

Додаток Г

```
import numpy as np
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.preprocessing import MinMaxScaler
import json

#генерація
np.random.seed(42)
n = 2000

cpu_temp  = np.random.uniform(20, 90, n)
env_temp  = np.random.uniform(15, 45, n)
load_pct  = np.random.uniform(0, 100, n)

def get_mode(t, l):
    if t < 45 and l < 40:   return 0  # NORMAL
    elif t < 60 and l < 70: return 1  # WARNING
    elif t < 75 and l < 90: return 2  # HIGH
    else:                   return 3  # EMERGENCY

#регресія
def get_cooling(t, l, e):
    base = (t - 20) / 70 * 70 + l * 0.25 + (e - 15) / 30 * 5
    return float(np.clip(base, 0, 100))

modes    = np.array([get_mode(t, l)        for t, l     in zip(cpu_temp, load_pct)])
```

```
cooling  = np.array([get_cooling(t, l, e)   for t, l, e  in zip(cpu_temp, load_pct,
env_temp)])


X = np.column_stack([cpu_temp, env_temp, load_pct])


#нормализація
scaler = MinMaxScaler(feature_range=(0, 1))
scaler.fit([[20, 15, 0], [90, 45, 100]])   # фиксированный диапазон
X_norm = scaler.transform(X)


#навчання класиф-ра
clf = MLPClassifier(
   hidden_layer_sizes=(8, 6),
   activation='tanh',
   max_iter=1000,
   random_state=42
)
clf.fit(X_norm, modes)
print(f"Classifier accuracy with Tanh: {clf.score(X_norm, modes):.3f}")


def export_weights(model, filename):
   layers = []
   for i, (W, b) in enumerate(zip(model.coefs_, model.intercepts_)):
      layers.append({
         "weights": W.tolist(),
         "biases":  b.tolist()
      })
```

```
    with open(filename, 'w') as f:
        json.dump(layers, f, indent=2)
    print(f"Saved {filename}")

export_weights(clf, "classifier_weights.json")

def to_c_array(name, arr):
    flat = np.array(arr).flatten()
    vals = ", ".join(f"{v:.6f}f" for v in flat)
    rows, *cols = np.array(arr).shape
    if cols:
        return f"const float {name}[{rows}][{cols[0]}] = {{{vals}}};"
    return f"const float {name}[{rows}] = {{{vals}}};"

layers = list(zip(clf.coefs_, clf.intercepts_))
for i, (W, b) in enumerate(layers):
    print(to_c_array(f"W{i+1}", W.T.tolist()))   # для удобства
    print(to_c_array(f"b{i+1}", b.tolist()))
```

Додаток Д

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <ESP32Servo.h>

#include "ai_model.h"

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

#define DHTPIN 15
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

#define ONE_WIRE_BUS 4
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

```
#define POT_PIN 34

#define SERVO_PIN 18
#define RELAY_PIN 23

Servo fanServo;

enum SystemState {
   STATE_NORMAL = 0,
   STATE_WARNING = 1,
   STATE_HIGH = 2,
   STATE_EMERGENCY = 3
};

SystemState currentState = STATE_NORMAL;

unsigned long previousMillis = 0;
const long interval = 1000;

struct Statistics {
   unsigned long total_runtime_ms = 0;
   float avg_cpu_temp = 0.0f;
   float max_cpu_temp = 25.0f;
```

```
    float min_cpu_temp = 50.0f;
    int state_changes = 0;
    int relay_activations = 0;
} stats;

SystemState previous_state = STATE_NORMAL;

void setup() {
    Serial.begin(115200);
    delay(2000);

    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
        Serial.println(F("SSD1306 initialization failed!"));
        for(;;);
    }
    display.clearDisplay();
    display.setTextColor(WHITE);
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.println("Initializing...");
    display.display();

    //Ініціалізація датчиків
```

```
dht.begin();
sensors.begin();

//Ініціалізація виконавчих пристроїв
pinMode(RELAY_PIN, OUTPUT);
digitalWrite(RELAY_PIN, LOW);

fanServo.attach(SERVO_PIN);
fanServo.write(0);

//лог
Serial.println(
    "Time(ms),"
    "CPU_Temp(C),"
    "Env_Temp(C),"
    "Load(%),"
    "AI_Cooling_Level(%),"
    "AI_Mode,"
    "Expected_Mode,"
    "Confidence,"
    "Fan_Angle(deg),"
    "Relay_State,"
    "State_Name"
```

```
    );

    previousMillis = millis();
}


int getExpectedMode(float cpu_temp, float load_percent) {
    if (cpu_temp < 45.0f && load_percent < 40.0f) {
        return 0;  // NORMAL
    }
    else if (cpu_temp < 60.0f && load_percent < 70.0f) {
        return 1;  // WARNING
    }
    else if (cpu_temp < 75.0f && load_percent < 90.0f) {
        return 2;  // HIGH
    }
    else {
        return 3;  // EMERGENCY
    }
}

const char* getStateName(SystemState state) {
    switch(state) {
```

```
        case STATE_NORMAL:    return "NORMAL";
        case STATE_WARNING:   return "WARNING";
        case STATE_HIGH:      return "HIGH";
        case STATE_EMERGENCY: return "EMERGENCY";
        default:              return "UNKNOWN";
    }
}

void loop() {
    unsigned long currentMillis = millis();

    if (currentMillis - previousMillis >= interval) {
        previousMillis = currentMillis;
        stats.total_runtime_ms += interval;

        sensors.requestTemperatures();
        float cpuTemp = sensors.getTempCByIndex(0);
        float envTemp = dht.readTemperature();

        int rawLoad = analogRead(POT_PIN);
        float loadPercent = (rawLoad / 4095.0) * 100.0;

        // Імітація нагріву
```

```
cpuTemp += (loadPercent / 100.0) * 65.0;

// Обробка помилок
if (isnan(cpuTemp) || cpuTemp < -20) cpuTemp = 35.0;
if (isnan(envTemp)) envTemp = 24.0;

// Фільтрування
static float filteredCpuTemp = 35.0;
filteredCpuTemp = 0.7f * filteredCpuTemp + 0.3f * cpuTemp;
cpuTemp = filteredCpuTemp;

// Статистика
stats.max_cpu_temp = max(stats.max_cpu_temp, cpuTemp);
stats.min_cpu_temp = min(stats.min_cpu_temp, cpuTemp);
stats.avg_cpu_temp = 0.95f * stats.avg_cpu_temp + 0.05f * cpuTemp;

//Інференс
AI_Output ai_result = predict_system_control(cpuTemp, envTemp,
loadPercent);

currentState = (SystemState)ai_result.system_mode;
float aiCoolingPercent = ai_result.cooling_level;
```

```
// Очікуваний режим
int expectedMode = getExpectedMode(cpuTemp, loadPercent);

// Відстеження змін
if (currentState != previous_state) {
    stats.state_changes++;
    previous_state = currentState;
}

float fanAngle = (aiCoolingPercent / 100.0) * 180.0;
fanAngle = constrain(fanAngle, 0, 180);
fanServo.write((int)fanAngle);

bool relayState = ai_result.relay_on;
digitalWrite(RELAY_PIN, relayState ? HIGH : LOW);

if (relayState) {
    stats.relay_activations++;
}

Serial.print(currentMillis); Serial.print(",");
Serial.print(cpuTemp, 1); Serial.print(",");
Serial.print(envTemp, 1); Serial.print(",");
```

```
Serial.print(loadPercent, 1); Serial.print(",");
Serial.print(aiCoolingPercent, 1); Serial.print(",");
Serial.print(ai_result.system_mode); Serial.print(",");
Serial.print(expectedMode); Serial.print(",");
Serial.print(ai_result.prediction_confidence, 2); Serial.print(",");
Serial.print((int)fanAngle); Serial.print(",");
Serial.print(relayState ? 1 : 0); Serial.print(",");
Serial.println(getStateName(currentState));

display.clearDisplay();
display.setTextSize(1);
display.setCursor(0, 0);

display.print("STATE: ");
display.print(getStateName(currentState));
display.print(" [");
display.print(stats.total_runtime_ms / 1000);
display.println("s]");
display.drawLine(0, 9, 128, 9, WHITE);

display.setCursor(0, 14);
display.print("T:"); display.print(cpuTemp, 1);
display.print("C  Load:"); display.print(loadPercent, 0);
```

```
        display.println("%");

        display.print("Cool:"); display.print(aiCoolingPercent, 0);
        display.print("%  Fan:");
        display.print((int)fanAngle); display.println("d");

        display.print("Conf:"); display.print(ai_result.prediction_confidence, 2);
        display.print("  Relay:");
        display.println(relayState ? "ON" : "OFF");

        display.print("Expected:");
        display.print(expectedMode);
        display.print(" Actual:");
        display.println(ai_result.system_mode);

        display.display();
    }
}
```

Додаток Е

```
#ifndef AI_MODEL_H
#define AI_MODEL_H

#include <math.h>
#include <Arduino.h>

struct AI_Output {
   float cooling_level;         //Рівень охолодження
   int   system_mode;
   float prediction_confidence;  // Впевненість
   bool  relay_on;              //реле
};

// нормалізація

static const float FEAT_MIN[3] = {20.0f, 15.0f,   0.0f};
static const float FEAT_MAX[3] = {90.0f, 45.0f, 100.0f};

inline float normalize_feature(float v, int idx) {
   float n = (v - FEAT_MIN[idx]) / (FEAT_MAX[idx] - FEAT_MIN[idx]);
   return fmaxf(0.0f, fminf(1.0f, n));
}
```

```
//Шар 1

static const float W1[8][3] = {
    { 0.058105f,  0.090687f, -1.933437f},
    { 2.719089f, -0.027419f, -2.382404f},
    { 1.173240f, -0.015615f,  0.865758f},
    { 1.736180f, -0.029136f, -2.110372f},
    {-1.748293f,  0.183695f, -0.334722f},
    {-1.462952f, -0.082895f, -1.203575f},
    {-1.329620f, -0.106020f, -0.779441f},
    { 2.711059f,  0.077725f, -2.030526f}
};

static const float b1[8] = {
    -0.095533f,  1.140434f, -0.643993f,  0.998733f,
     0.826276f, -0.021071f,  0.926350f, -1.002216f
};

//Шар 2

static const float W2[6][8] = {
```

```
   { 0.142989f,  1.054426f,  1.865686f,  1.929940f, -3.870746f,  0.247094f, -
2.430027f,  2.571461f},

   {-0.372239f, -1.797761f,  1.159867f, -0.924899f, -1.505500f, -1.390698f, -
0.477126f,  2.226732f},

   { 2.037408f,  0.863223f, -1.964546f,  2.823067f,  0.057288f,  0.900144f,
1.609909f,  0.265866f},

   { 0.680563f, -0.967106f,  1.387854f, -0.668303f, -1.634567f, -0.266810f, -
1.621750f,  0.863339f},

   {-0.652883f, -2.174432f,  1.181807f, -0.507019f, -1.396960f, -1.356433f, -
1.765637f,  1.988072f},

   {-0.511014f,  2.248498f, -1.922482f,  2.234952f,  1.572270f,  0.732315f,
2.048425f, -2.565113f}

};


static const float b2[6] = {

   -0.053734f,  0.662742f, -0.126273f, -0.672592f,  0.330868f, -0.211652f

};


//Шар 3


static const float W3[4][6] = {

   {-3.650250f, -3.101664f,  4.983327f, -0.975332f, -2.035993f, -0.000525f},

   { 2.885091f, -3.233339f, -2.331170f, -2.862671f, -3.742679f,  2.774371f},

   { 0.470335f,  2.384401f, -0.557148f, -1.382297f,  3.158601f,  3.807211f},

   { 0.073151f,  2.478016f, -0.851497f,  3.025966f,  1.844448f, -6.346993f}

};
```

```
static const float b3[4] = {
   -1.254420f, -0.621579f,  0.601235f,  1.192072f
};


// Поріг увімкнення
static const float RELAY_ON_CONFIDENCE  = 0.90f;

// Поріг вимкнення
static const float RELAY_OFF_CONFIDENCE = 0.75f;

// Мін. нормалізована температура для увімкнення
static const float RELAY_ON_TEMP_NORM   = 0.70f;

AI_Output predict_system_control(float cpu_temp, float env_temp, float load_pct)
{
   AI_Output out;

   //Нормалізація вхідного вектора
   float x[3] = {
      normalize_feature(cpu_temp, 0),
      normalize_feature(env_temp, 1),
```

```
    normalize_feature(load_pct, 2)
};

//Перший прихований шар
float h1[8];
for (int i = 0; i < 8; i++) {
    float z = b1[i];
    for (int j = 0; j < 3; j++) z += W1[i][j] * x[j];
    h1[i] = tanhf(z);
}

// Другий прихований шар
float h2[6];
for (int i = 0; i < 6; i++) {
    float z = b2[i];
    for (int j = 0; j < 8; j++) z += W2[i][j] * h1[j];
    h2[i] = tanhf(z);
}

//Вихідний шар
float logits[4];
for (int i = 0; i < 4; i++) {
    logits[i] = b3[i];
```

```
    for (int j = 0; j < 6; j++) logits[i] += W3[i][j] * h2[j];
}

// Softmax
float max_l = logits[0];
for (int i = 1; i < 4; i++)
    if (logits[i] > max_l) max_l = logits[i];

float probs[4], sum_e = 0.0f;
for (int i = 0; i < 4; i++) {
    probs[i] = expf(logits[i] - max_l);
    sum_e   += probs[i];
}

int best = 0;
for (int i = 0; i < 4; i++) {
    probs[i] /= sum_e;
    if (probs[i] > probs[best]) best = i;
}

out.system_mode          = best;
out.prediction_confidence = probs[best];
```

```
    //Рівень охолодження
    static const float BASE_COOLING[4] = {10.0f, 40.0f, 70.0f, 100.0f};

    float extra = x[0] * 15.0f   // головний пріоритет темп.
                + x[2] * 10.0f   // реакція на навантаження
                + x[1] *  5.0f;  // облік температури оточення

    out.cooling_level = BASE_COOLING[best] + (extra *
out.prediction_confidence);
    out.cooling_level = fmaxf(0.0f, fminf(100.0f, out.cooling_level));

    //гістерезис
    static bool relay_state_prev = false;

    bool relay_new;

    if (!relay_state_prev) {
        relay_new = (out.system_mode == 3) &&
                    (out.prediction_confidence > RELAY_ON_CONFIDENCE) &&
                    (x[0] > RELAY_ON_TEMP_NORM);
    }
    else {
        //Реле увімкнено
```

```
        relay_new = !( (out.system_mode < 2) ||
                       (out.system_mode == 2 && out.prediction_confidence <
RELAY_OFF_CONFIDENCE) );
    }

    relay_state_prev = relay_new;
    out.relay_on     = relay_new;

    return out;
}

#endif // AI_MODEL_H
```