Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

за спеціальністю 123 «Комп'ютерна інженерія»

Тема наукової роботи: МЕТОД ДИСТАНЦІЙНОГО КЕРУВАННЯ ЕНЕРГОЕФЕКТИВНІСТЮ БУДИНКУ ЗА ДОПОМОГОЮ ІОТ

Виконав _____ Р.М. Котенко

Керівник роботи _____ Ю.О. Кумченко

Консультант _____ Ю.О. Кумченко

Нормоконтроль _____ Д. І. Кузнєцов

Завідувач кафедри _____ А. І. Купін

Кривий Ріг
2025

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь вищої освіти магістр
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

"____" _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

_____
(прізвище, ім'я, по батькові)

1. Тема роботи _____
_____
_____

керівник роботи _____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "___"_____ 20__року №__

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи _____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____
_____
_____
_____
_____
_____

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота магістра: 108 с., 15 рис., 4 табл., 2 додатки, 25 джерел.

Об'єкт дослідження – процес споживання та розподілу енергетичних ресурсів у житловій будівлі.

Мета роботи – підвищення енергоефективності та енергетичної стійкості житлового будинку шляхом розробки адаптивної моделі керування на базі технологій Інтернету речей.

Методи дослідження: теорія нечіткої логіки, системний аналіз, математичне моделювання, імітаційний експеримент.

У роботі розроблено математичну модель нечіткого керування системою опалення, яка, на відміну від існуючих аналогів, комплексно враховує прогноз погоди та графіки відключень електроенергії. Спроектовано архітектуру автономної IoT-системи за принципом "Local-First" на базі мікроконтролерів ESP32 та одноплатного комп’ютера Raspberry Pi 4. Розроблено програмне забезпечення для реалізації шлюзу керування та інтеграції з гібридним сонячним інвертором. Експериментально доведено, що запропонована модель дозволяє знизити добове споживання електроенергії на 18.8% та забезпечити у 4 рази тривалішу роботу системи опалення від резервного акумулятора під час блекаутів порівняно з класичними термостатами.

Ключові слова: ЕНЕРГОЕФЕКТИВНІСТЬ, ІНТЕРНЕТ РЕЧЕЙ, НЕЧІТКА ЛОГІКА, РОЗУМНИЙ БУДИНОК, MQTT, RASPBERRY PI, ESP32.

Master's qualification thesis: 108 pp., 15 fig., 4 tabl., 2 appendices, 25 references.

The object of research is the process of energy resources consumption and distribution in a residential building.

The purpose of the work is to increase energy efficiency and energy resilience of a residential building by developing an adaptive control model based on Internet of Things technologies.

Research methods: fuzzy logic theory, system analysis, mathematical modeling, simulation experiment.

The thesis develops a mathematical model of fuzzy heating control which, unlike existing analogues, comprehensively considers weather forecasts and power outage schedules. The architecture of an autonomous IoT system based on the "Local-First" principle using ESP32 microcontrollers and a Raspberry Pi 4 single-board computer is designed. Software for implementing the control gateway and integration with a hybrid solar inverter has been developed. It is experimentally proven that the proposed model allows reducing daily electricity consumption by 18.8% and ensuring 4 times longer operation of the heating system from a backup battery during blackouts compared to classic thermostats.

Keywords: ENERGY EFFICIENCY, INTERNET OF THINGS, FUZZY LOGIC, SMART HOME, MQTT, RASPBERRY PI, ESP32.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ДБЖ — Джерело безперебійного живлення

ККД — Коефіцієнт корисної дії

ОЕС — Об'єднана енергосистема України

ОС — Операційна система **ПЗ** — Програмне забезпечення

ШІМ — Широтно-імпульсна модуляція (див. PWM)

API — Application Programming Interface (Програмний інтерфейс застосунку)

BAS — Building Automation Systems (Системи автоматизації будівель)

BEMS — Building Energy Management Systems (Системи управління енергією в будівлях)

BLE — Bluetooth Low Energy (Bluetooth з низьким енергоспоживанням)

CoG — Center of Gravity (Метод центру ваги для дефазифікації)

DDC — Direct Digital Control (Пряме цифрове керування)

DIY — Do It Yourself ("Зроби сам", аматорський підхід)

ESP — Серія мікроконтролерів компанії Espressif Systems

FDD — Fault Detection and Diagnostics (Виявлення та діагностика несправностей)

FLC — Fuzzy Logic Control (Керування на основі нечіткої логіки)

GPIO — General Purpose Input/Output (Інтерфейс введення/виведення загального призначення)

HA — Home Assistant (Платформа для автоматизації)

HVAC — Heating, Ventilation, and Air Conditioning (Опалення, вентиляція та кондиціонування)

I2C — Inter-Integrated Circuit (Послідовна шина даних)

IoT — Internet of Things (Інтернет речей)

LED — Light-Emitting Diode (Світлодіод)

MPC — Model Predictive Control (Модельне предиктивне керування)

MQTT — Message Queuing Telemetry Transport (Протокол передачі повідомлень)

PID — Proportional-Integral-Derivative (Пропорційно-інтегрально-диференціальний регулятор)

PWM — Pulse-Width Modulation (Широтно-імпульсна модуляція)

RMSE — Root Mean Square Error (Середньоквадратичне відхилення)

SBC — Single-Board Computer (Одноплатний комп'ютер)

SOC — State of Charge (Рівень заряду акумуляторної батареї)

SSR — Solid State Relay (Твердотільне реле)

TTL — Transistor-Transistor Logic (Транзисторно-транзисторна логіка)

UART — Universal Asynchronous Receiver-Transmitter (Універсальний асинхронний приймач-передавач)

WDT — Watchdog Timer (Сторожовий таймер)

YAML — YAML Ain't Markup Language (Мова серіалізації даних)

ВСТУП

Ефективне використання енергетичних ресурсів є однією з ключових проблем сучасної світової економіки, проте для України це питання в останні роки набуло безпрецедентної гостроти та стратегічного значення. Нормативно-правовою основою для впровадження енергоефективних заходів є Закон України «Про енергетичну ефективність будівель» [1] та Національний план дій з енергоефективності на період до 2030 року [2]. По-перше, це висока та нестабільна вартість енергоносіїв. Україна, навіть до повномасштабного вторгнення, знаходилась у процесі інтеграції до європейського енергетичного ринку, що передбачало неминуче зростання внутрішніх цін на газ та електроенергію. В умовах поточної складної економічної ситуації, спричиненої війною, непомірні комунальні витрати стають критичним навантаженням як для домогосподарств, так і для державного бюджету, що змушений покривати величезні субсидійні дефіцити. У 2024-2025 роках відбулися чергові етапи підвищення тарифів, що робить задачу персональної економії не просто бажаною, а економічно необхідною для виживання.

По-друге, найвагомішим фактором актуальності є цілеспрямовані терористичні атаки агресора (росії) на енергетичну інфраструктуру України. Починаючи з 2022 року, ворог систематично завдає масованих комбінованих ударів по об'єктах генерації та розподілу, уражаючи теплові (ТЕС), гідроелектростанції (ГЕС) та ключові підстанції "Укренерго". Атаки агресора на енергосистему призвели до катастрофічних наслідків:

- **Критична дестабілізація Об'єднаної енергосистеми (ОЕС)**, що виражається у вимушених стабілізаційних та аварійних відключеннях (блекаутах) для збалансування мережі.
- **Руйнування систем централізованого теплопостачання** у багатьох регіонах (як-от у Харкові, Охтирці, частково в Києві та інших містах), що змушує населення масово переходити на автономне електричне опалення (конвектори, бойлери), створюючи вибухові пікові навантаження на і без того пошкоджену електромережу.
- **Фізичне знищення значної частини маневрової генерації**, що створює перманентний дефіцит електроенергії в системі, особливо у пікові ранкові та вечірні години.

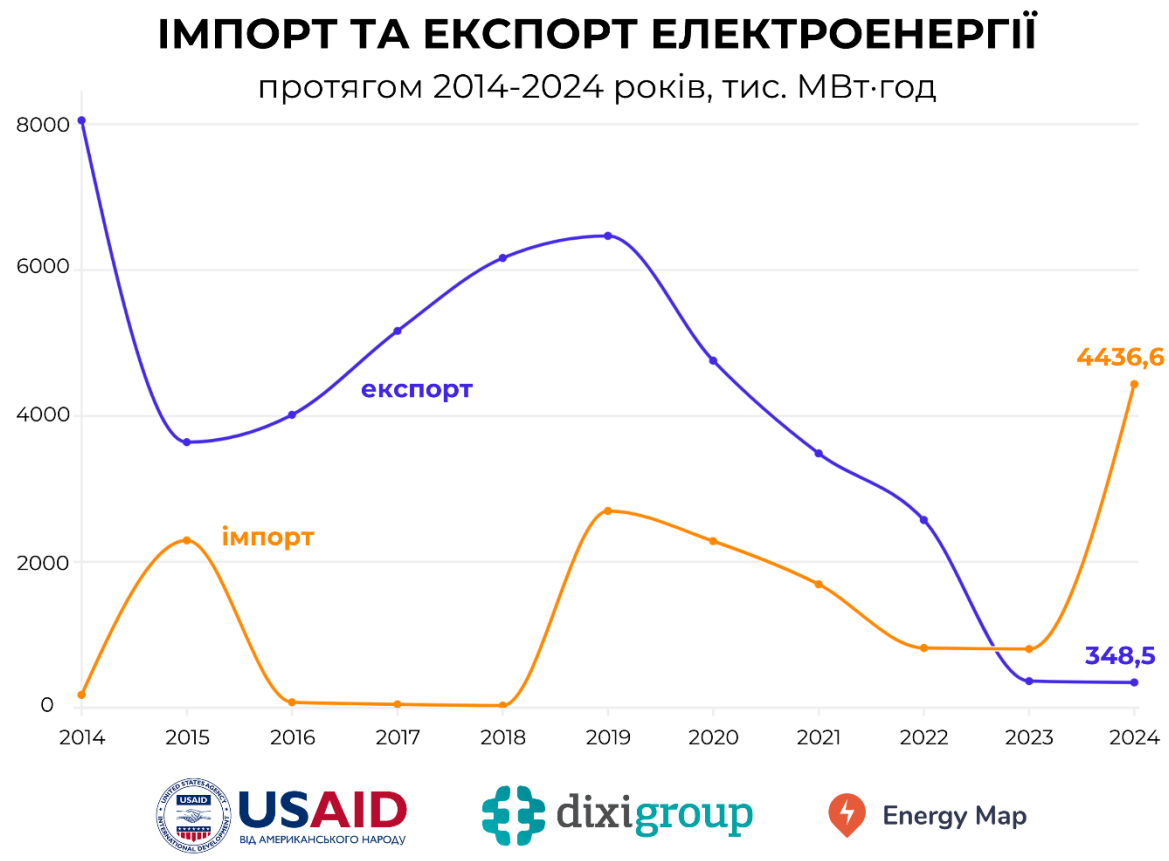


Рисунок 1.1 Динаміка імпорту та експорту електроенергії 2014-2024 рр.

Як бачимо на графіку імпорту\експорту електроенергії в Укріїні критично не вистачає власної генерації. (Рисунок 1.1)

В таких екстремальних умовах проблема енергоефективності трансформується з суто економічної (економія грошей) у **проблему національної безпеки та енергетичної стійкості (resilience)**. Кожен зекономлений домогосподарством кіловат енергії — це не лише зменшення фінансового навантаження, але й пряме зниження дефіциту в ОЕС, що допомагає уникнути колапсу мережі та скоротити тривалість відключень для мільйонів інших громадян.

По-третє, це технологічна застарілість житлового фонду. Більшість будівель в Україні були зведені за старими будівельними нормами і мають колосальні тепловтрати. Водночас традиційні системи керування будівлями (Building Energy Management Systems, BEMS) є занадто дорогими, складними та закритими для масового впровадження у приватному секторі.

Ідея роботи полягає у відході від реактивних (ввімк/вимкн) та статичних (за розкладом) методів керування до **проактивної, адаптивної моделі**. Така модель, на відміну від існуючих, має враховувати не лише поточні показники

сенсорів, але й зовнішні фактори (прогноз погоди, наявність електроенергії в мережі) та патерни поведінки мешканців для прийняття оптимальних рішень на випередження.

Отже, **актуальність даної роботи** полягає у вирішенні нагального науково-практичного завдання: розробці доступної та гнучкої моделі керування енергоефективністю будинку на базі IoT, що здатна підвищити не лише економічну ефективність, але й енергетичну стійкість домогосподарств в умовах дефіцитної та пошкодженої енергосистеми України.

Метою кваліфікаційної роботи магістра є підвищення енергоефективності житлового будинку шляхом розробки та дослідження адаптивної моделі керування виконавчими пристроями (системами опалення та освітлення) на основі даних, отриманих з мережі IoT-сенсорів.

Для досягнення поставленої мети необхідно вирішити такі **завдання дослідження**:

1. Провести системний аналіз існуючих підходів та систем керування енергоефективністю (BEMS), а також математичних моделей, що в них використовуються.
2. Дослідити сучасні апаратні та програмні платформи Інтернету речей (IoT), протоколи передачі даних та їх придатність для вирішення завдань енергомоніторингу та керування.
3. Виявити ключові недоліки та "протиріччя" існуючих рішень стосовно умов України та обґрунтувати напрямок дослідження.
4. Розробити математичну модель (наприклад, на основі нечіткої логіки або предиктивного аналізу) для прийняття рішень щодо керування енергоспоживанням.
5. Спроектувати архітектуру інформаційної IoT-системи для збору даних, реалізації моделі та взаємодії з виконавчими пристроями.
6. Розробити програмне забезпечення для ключових вузлів системи, що реалізує запропоновану модель.
7. Провести експериментальне дослідження ефективності запропонованої моделі шляхом комп'ютерного моделювання та порівняти її з базовими сценаріями керування.

Об'єкт дослідження – процес споживання та розподілу енергетичних ресурсів (електроенергії, тепла) у житловій будівлі.

Предмет дослідження – моделі, методи та алгоритми керування енергоефективністю будинку на основі технологій Інтернету речей.

Методи дослідження. Для досягнення поставленої мети використовувалися методи теорії автоматичного керування, теорії нечітких множин та нечіткої

логіки, методи математичного та імітаційного моделювання, системний аналіз, теорія комп'ютерних мереж та протоколів IoT, методи об'єктно-орієнтованого програмування.

Наукова новизна одержаних результатів полягає у наступному:

1. Запропоновано модель керування енергоефективністю, що, на відміну від існуючих, комплексно враховує три типи вхідних даних: показники внутрішніх сенсорів (температура, присутність), зовнішні фактори (прогноз погоди, графіки відключень) та прості патерни поведінки мешканців.
2. Удосконалено метод прийняття рішень на основі нечіткої логіки, що, шляхом введення динамічних вагових коефіцієнтів для правил, дозволяє моделі гнучко змінювати пріоритет (наприклад, між "комфортом" та "економією") залежно від наявності централізованого енергопостачання.

Практичне значення одержаних результатів. Розроблена модель та архітектура системи можуть бути використані для створення доступних комерційних або "DIY" (Do It Yourself) рішень для модернізації житлових будинків та квартир. Впровадження системи дозволяє (за результатами моделювання) знизити споживання енергії на опалення до 15-20% при одночасному підвищенні комфорту та забезпеченні енергетичної стійкості під час відключень.

РОЗДІЛ 1. ОГЛЯД ТА КЛАСИФІКАЦІЯ СИСТЕМ КЕРУВАННЯ ЕНЕРГІЄЮ В БУДІВЛЯХ (BEMS, BAS)

1.1.1 Визначення, функції та актуальність

Сучасні будівлі, від приватних житлових будинків до великих комерційних та промислових комплексів, є складними інженерними системами. Ефективне управління цими системами є критично важливим для забезпечення комфорту, безпеки та раціонального використання ресурсів. У цьому контексті ключову роль відіграють системи автоматизації та управління. Для коректного аналізу предметної області необхідно чітко розмежувати два фундаментальних поняття: BAS та BEMS.

Системи автоматизації будівель (BAS, Building Automation Systems), часто відомі як "системи інтелектуальних будівель", є ширшим поняттям **[4]**. Це комплексні апаратно-програмні рішення, призначені для централізованого моніторингу та автоматизованого керування різноманітними інженерними підсистемами будівлі. До типових підсистем, якими керує BAS, належать:

- Системи опалення, вентиляції та кондиціонування (HVAC або ОВіК);
- Системи освітлення (внутрішнього та зовнішнього);
- Системи контролю доступу та безпеки (СКУД);
- Системи пожежної сигналізації;
- Керування ліфтовим господарством та іншими механізмами.

Головна мета BAS полягає в оптимізації взаємодії цих систем, підвищенні рівня комфорту та безпеки для мешканців, а також зниженні експлуатаційних витрат.

Системи управління енергією в будівлях (BEMS, Building Energy Management Systems) є більш вузькоспеціалізованим класом систем, хоча їхні функції часто інтегровані в сучасні BAS. BEMS цілеспрямовано фокусується на **моніторингу, контролі та оптимізації** саме енергоспоживання будівлі. Якщо головна мета BAS – комфорт та безпека, то головна мета BEMS – енергоефективність та зниження витрат на енергоносії. BEMS збирає дані з лічильників (електроенергії, тепла, газу, води), аналізує їх у співвідношенні з даними про умови експлуатації (погода, присутність людей, графіки роботи) та застосовує алгоритми керування для мінімізації споживання.

Актуальність впровадження таких систем, особливо в Україні, обумовлена не лише глобальними тенденціями сталого розвитку та вимогами європейських директив з енергоефективності (наприклад, EPBD - Energy Performance of Buildings Directive) **[3]**, але й критичною необхідністю раціонального використання ресурсів в умовах дефіцитної енергосистеми та стрімкого зростання вартості енергоносіїв.

1.1.2 Історичний розвиток систем автоматизації

Розуміння сучасного стану BEMS/BAS неможливе без аналізу їх еволюції, що налічує понад століття розвитку технологій керування.

- **Фаза 1: Локальні механічні та пневматичні системи (початок-середина XX ст.).** Перші спроби автоматизації були суто механічними. Класичним прикладом є винахід кімнатного термостата. У великих комерційних будівлях середини XX століття домінували пневматичні системи керування, де для передачі сигналів та живлення виконавчих механізмів (клапанів) використовувалося стиснене повітря. Ці системи були надійними, але громіздкими, неточними та не мали жодної централізації.

- **Фаза 2: Електромеханічні та аналогові електронні системи (1950-ті – 1980-ті).** З розвитком електроніки пневматику почали замінювати електромеханічні реле та прості аналогові контролери. З'явилися перші централізовані пульти управління, що дозволяли оператору бачити стан обладнання та вручну вмикати/вимикати його. Керування базувалося на простих таймерах та релейній логіці.
- **Фаза 3: Поява цифрових контролерів (DDC) (1980-ті – 1990-ті).** Справжня революція в автоматизації будівель відбулася з появою мікропроцесорів та **DDC (Direct Digital Control)** контролерів. Це дозволило вперше реалізувати складні алгоритми керування (наприклад, ПІД-регулювання), задавати гнучкі розклади та отримувати зворотний зв'язок від цифрових датчиків. Системи стали програмно-керованими. Однак на цьому етапі вони були переважно **пропрієтарними** – кожен виробник (Johnson Controls, Honeywell, Siemens) використовував власні закриті протоколи, що "прив'язувало" клієнта до одного постачальника.
- **Фаза 4: Мережева інтеграція та відкриті протоколи (кінець 1990-х – 2010-ті).** Головною проблемою попереднього етапу була неможливість інтеграції обладнання різних виробників. Це призвело до появи стандартизованих **відкритих протоколів**. Найбільшого поширення набули **BACnet** (розроблений під егідою ASHRAE, став домінуючим у США та в сегменті HVAC) та **LonWorks** (розроблений Echelon Corporation, популярний в Європі). Це дозволило будувати інтегровані системи, де контролери HVAC, освітлення та безпеки могли обмінюватися даними через загальну мережу (зазвичай LAN).
- **Фаза 5: Сучасні інтелектуальні та IoT-системи (2010-ті – наш час).** Останній етап характеризується глибокою інтеграцією BAS/BEMS з IT-інфраструктурою. Системи отримали веб-інтерфейси, почали використовувати хмарні платформи для аналітики та віддаленого доступу. Паралельно розвиток **Інтернету речей (IoT)** [12] спричинив появу масових, дешевих та бездротових сенсорів і виконавчих пристроїв, що стало основою для систем "Розумного будинку" та новітнього покоління гнучких BEMS.

1.1.3 Архітектура та основні компоненти BEMS/BAS

Класична архітектура BEMS/BAS традиційно описується у вигляді ієрархічної піраміди, що складається з трьох основних рівнів. (Рисунок 1.2)

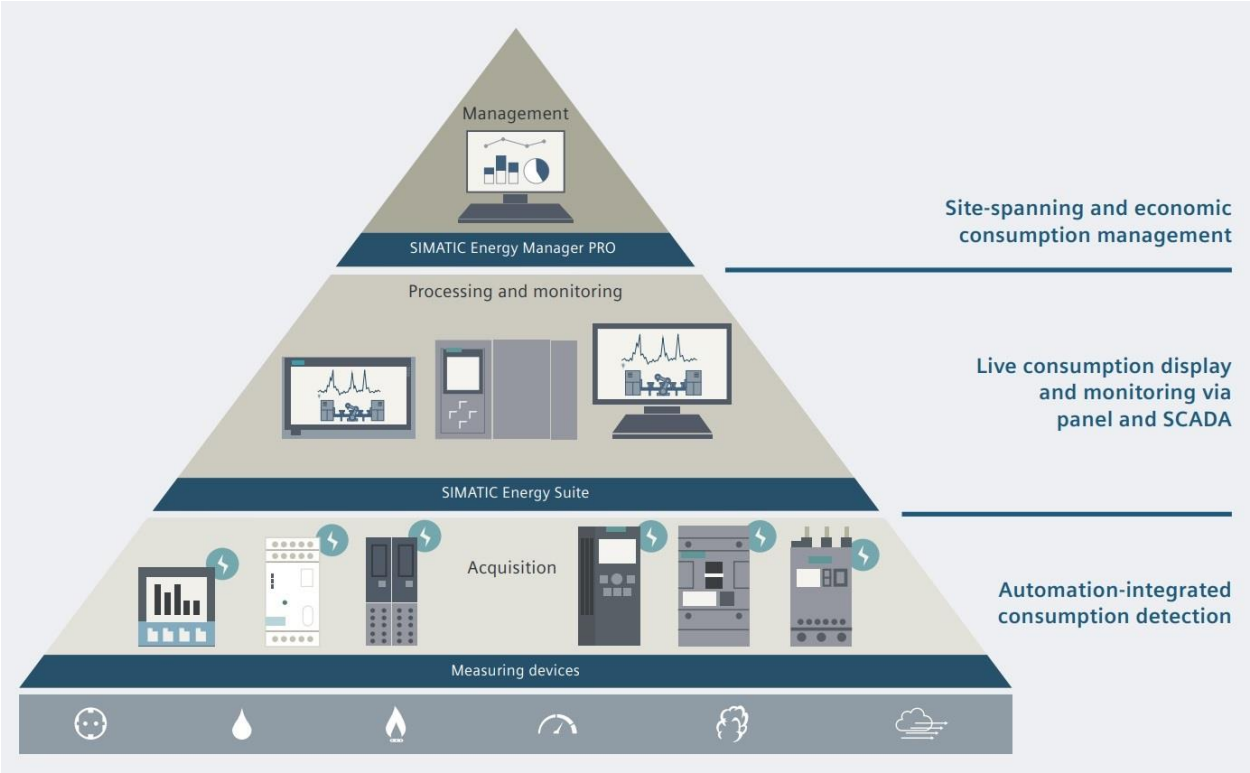


Рисунок 1.2 Класична архітектура BEMS/BAS

1. **Рівень 0/1: Польовий рівень (Field Level).** Це "очі, вуха та руки" системи, що безпосередньо взаємодіють з фізичним середовищем та інженерним обладнанням.
   - **Сенсори (Датчики):** Пристрої для збору даних. До них належать датчики температури, вологості, тиску, рівня CO2, присутності (PIR-сенсори), освітленості, а також лічильники та вимірювачі споживання (електроенергії, тепла, газу).
   - **Виконавчі механізми (Actuators):** Пристрої, що виконують керуючі команди. Це сервоприводи клапанів (для систем опалення), приводи повітряних заслінок (для вентиляції), реле та димери (для освітлення), частотні перетворювачі (для керування швидкістю двигунів насосів та вентиляторів).

2. **Рівень 2: Рівень автоматизації (Automation Level).** Це "локальний мозок" системи.
   - **Контролери (DDC/PLC):** Це мікропроцесорні пристрої, які фізично підключені до сенсорів та виконавчих механізмів польового рівня. Вони містять програмну логіку (програми, алгоритми, розклади, уставки) та автономно виконують завдання керування для своєї зони (наприклад, окремий поверх, вентиляційна установка чи котельня). Цей рівень є критично

важливим для надійності, оскільки він продовжує працювати навіть у разі втрати зв'язку з верхнім рівнем.

3. **Рівень 3: Рівень управління (Management Level).** Це "центральний мозок" або диспетчерський пункт.
   - **Сервери та Робочі станції (HMI/SCADA):** Це комп'ютери зі спеціалізованим програмним забезпеченням. Вони збирають дані з усіх контролерів рівня автоматизації, архівують їх у базах даних, візуалізують процеси для операторів (графіки, мнемосхеми), обробляють тривоги (alarms) та дозволяють централізовано змінювати глобальні налаштування системи. Саме на цьому рівні зазвичай виконуються складні алгоритми оптимізації та глибока аналітика енергоспоживання (BEMS).

   Основні **функції**, які реалізує ця архітектура, можна згрупувати наступним чином:

- **Моніторинг:** Безперервний збір та реєстрація даних про стан інженерних систем та параметри середовища.
- **Керування:** Автоматичне підтримання заданих параметрів (уставок) за допомогою алгоритмів (напр., ПІД-регулювання) та виконання логіки за розкладом (напр., переведення будівлі в "нічний режим").
- **Оптимізація:** Застосування просунутих алгоритмів для зниження енергоспоживання. Приклади включають "оптимальний старт/стоп" (розрахунок точного часу запуску/зупинки системи опалення, щоб досягти заданої температури *до* початку робочого дня, а не підтримувати її всю ніч) або "керування піковим навантаженням" (тимчасове відключення некритичних споживачів під час годин максимального попиту на електроенергію).
- **Аналітика та Звітність:** Виявлення та діагностика несправностей (FDD - Fault Detection and Diagnostics), аналіз тенденцій споживання, автоматичне формування звітів для енергоменеджерів.

1.1.4 Класифікація систем керування

Системи BEMS/BAS можна класифікувати за декількома ключовими ознаками, що відображають їх масштаб, технологічну базу та рівень інтелектуальності.

1. За масштабом об'єкта:

- **Системи "Розумний Дім" (Smart Home):** Призначені для приватних будинків та квартир. Пріоритетом є зручність користувача (комфорт,

віддалене керування зі смартфона, голосові асистенти). Енергоефективність часто є вторинною функцією.

- **Системи для малих та середніх комерційних будівель (Light Commercial BEMS):** Призначені для невеликих офісів, магазинів. Часто є масштабованими версіями "розумних будинків" або спрощеними версіями промислових BEMS.
- **Повномасштабні комерційні та промислові BEMS (Enterprise BEMS/BAS):** Призначені для великих офісних центрів, готелів, лікарень, аеропортів, промислових підприємств. Характеризуються високою складністю, тисячами точок моніторингу та керування, та високими вимогами до надійності.

2. За архітектурою та протоколами:

- **Пропрієтарні (закриті) системи:** Використовують апаратне та програмне забезпечення, а також протоколи зв'язку одного конкретного виробника. Це створює "vendor lock-in" – залежність клієнта від одного постачальника послуг та обладнання.
- **Відкриті системи:** Базуються на стандартизованих, відкритих протоколах. Це дозволяє інтегрувати в єдину систему обладнання та контролери від різних виробників, що дає гнучкість у проектуванні та модернізації. Найбільш поширеними відкритими протоколами в BEMS є:
  - **BACnet (Building Automation and Control Networks):** Домінуючий стандарт для комерційних будівель, особливо в сегменті ОВіК.
  - **LonWorks (Local Operating Network):** Мережева платформа, що часто використовується для освітлення, безпеки та інших підсистем.
  - **KNX:** Популярний світовий стандарт (особливо в Європі) для автоматизації житлових та легких комерційних будівель.
  - **Modbus:** Старий, але все ще широко розповсюджений промисловий протокол для зв'язку з лічильниками, частотними перетворювачами та іншим обладнанням.

3. За рівнем інтелектуальності (Покоління BEMS):

- **Перше покоління (Локальна автоматизація):** Автономні контролери, що працюють за жорстким розкладом.
- **Друге покоління (Централізоване керування):** Наявність центральної диспетчерської станції (HMI) для моніторингу та ручного коригування уставок.
- **Третє покоління (Інтегровані системи):** Системи, що об'єднують різні підсистеми (напр., ОВіК та освітлення) за допомогою відкритих протоколів для спільної роботи.

- **Четверте покоління (Інтелектуальні BEMS, i-BEMS):** Сучасне покоління систем, яке є ключовим для наукових досліджень. Вони активно використовують IT-технології:
  - **Хмарні платформи:** Для збору "великих даних" (Big Data) з багатьох будівель та їх глибокого аналізу.
  - **Штучний інтелект (AI) та Машинне навчання (ML):** Для побудови предиктивних (прогнозуючих) моделей. Система може прогнозувати споживання та внутрішню температуру на основі прогнозу погоди, історичних даних та патернів поведінки людей.
  - **Цифрові двійники (Digital Twins):** Створення детальної віртуальної моделі будівлі для симуляції різних сценаріїв керування перед їх реальним застосуванням.
  - **Інтеграція з IoT:** Використання дешевих бездротових сенсорів для отримання більш гранульованих даних (наприклад, температура в кожній окремій кімнаті, а не одна на весь поверх), що є відходом від традиційної дорогої "дротової" автоматизації.
  -

1.1.5 Висновки до підрозділу

Системи керування енергією та автоматизації будівель (BEMS/BAS) пройшли довгий шлях еволюції: від простих механічних пристроїв до складних, інтелектуальних програмно-апаратних комплексів. Вони є ключовим та безальтернативним інструментом для досягнення суттєвої енергоефективності у сучасному будівництві та експлуатації.

Аналіз показав чіткий технологічний тренд: перехід від статичних, закритих та реактивних систем до відкритих, розподілених та, найголовніше, **проактивних (прогнозуючих) та інтелектуальних** систем. Сучасні і-BEMS четвертого покоління, що використовують машинне навчання та хмарну аналітику, демонструють найвищу ефективність.

Однак, впровадження таких повномасштабних, промислових і-BEMS є надзвичайно дорогим та складним процесом, що робить їх недоступними для наймасовішого сегменту – існуючого житлового фонду та малих комерційних об'єктів. Водночас, паралельно розвивався ринок масових, доступних технологій "Розумного будинку" на базі IoT.

Це створює технологічний розрив: між потужними, але дорогими промисловими BEMS, та доступними, але функціонально обмеженими IoT-рішеннями. Саме аналізу недоліків цих масових IoT-систем та дослідженню можливостей їх вдосконалення для вирішення завдань енергоефективності присвячені наступні підрозділи даної роботи.

1.2 Аналіз ринку та недоліки масових IoT-рішень для "розумного будинку"

1.2.1 Перехід від BEMS до масового ринку "Smart Home"

Попередній підрозділ 1.1 продемонстрував, що професійні системи керування будівлями (BEMS) є потужними, комплексними, але водночас дорогими, складними в інсталяції та обслуговуванні. Вони історично орієнтовані на великий комерційний сектор (офісні центри, готелі, промислові об'єкти), що робить їх впровадження у приватному житловому фонді економічно невиправданим.

Паралельно, протягом останнього десятиліття, відбувся вибуховий розвиток ринку **"Розумного будинку" (Smart Home)**. Цей ринок став можливим завдяки стрімкому здешевленню мікроелектроніки, зокрема мікроконтролерів із вбудованими модулями бездротового зв'язку (Wi-Fi, Bluetooth), та повсюдному поширенню смартфонів як універсального пульта керування.

Між BEMS та Smart Home існує **фундаментальна відмінність у меті**. Якщо професійні BEMS розроблені з пріоритетом на **оптимізацію експлуатації** та повернення інвестицій (ROI) за рахунок економії ресурсів, то масовий ринок Smart Home історично фокусується на **комфорті та зручності** для кінцевого споживача (наприклад, голосове керування освітленням, віддалений контроль замків, автоматичне відкриття штор). Питання енергоефективності в них часто є вторинною, маркетинговою функцією, а не ядром системи.

1.2.2 Огляд ключових гравців та екосистем на ринку IoT

Сучасний ринок "розумного будинку" значною мірою поділений між кількома великими, переважно закритими, екосистемами та величезним сегментом масових "white-label" пристроїв.

1. Глобальні екосистеми ("Велика трійка"):

- **Google Home / Nest:** Ця екосистема, особливо завдяки придбанню компанії Nest, зробила значний крок у бік інтелектуального керування. Термостати Nest відомі своїми алгоритмами машинного навчання, що здатні вивчати звички мешканців та оптимізувати графіки опалення. Однак, вони залишаються дорогим рішенням, орієнтованим на ринок США/Європи, і їхня логіка є "чорною скринькою", що ускладнює кастомізацію. (Рисунок 1.3)

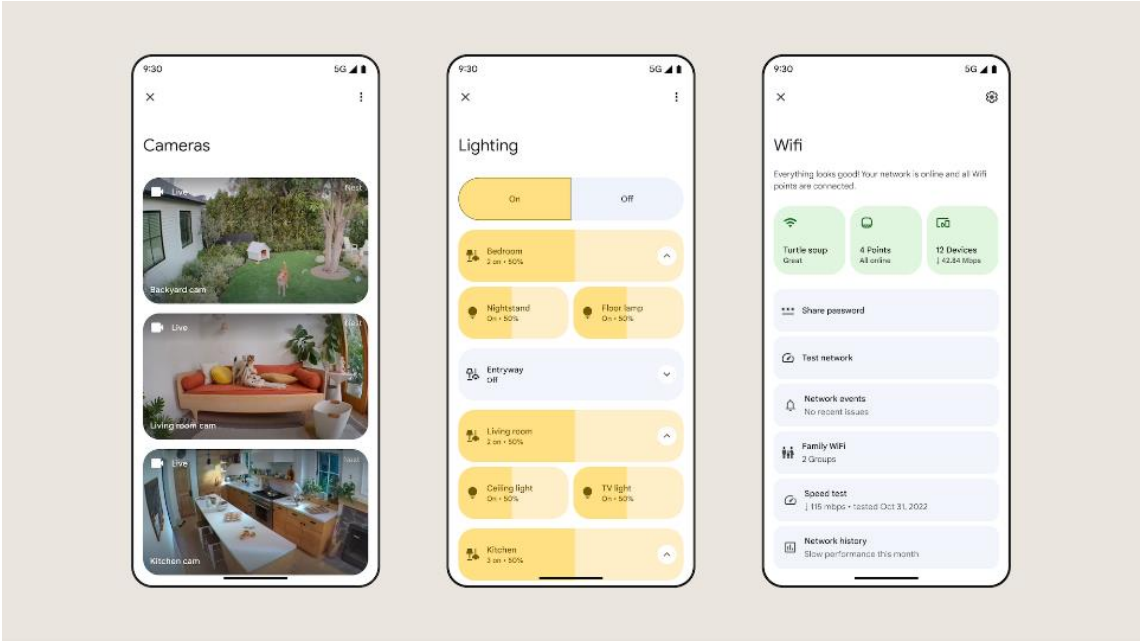


Рисунок 1.3 Інтерфейс екосистеми Google Home

- **Amazon Alexa:** Екосистема Amazon домінує за рахунок голосового асистента Alexa та гігантської кількості сумісних пристроїв від сторонніх виробників ("Works with Alexa"). Платформа є потужним агрегатором, але її власні інструменти автоматизації (Routines) є досить простими і не призначені для складної оптимізації. (Рисунок 1.4)

Рисунок 1.4 Пристрої екосистеми Amazon Alexa

- **Apple HomeKit:** Екосистема Apple традиційно робить головний акцент на **безпеці даних та приватності**, вимагаючи від виробників спеціальної сертифікації. Це створює надійну, але найбільш закриту та обмежену за вибором обладнання платформу.

Рисунок 1.5 — Інтерфейс Apple HomeKit

2. Масові платформи та DIY-рішення:

- **Tuya / Smart Life:** Це не бренд, а гігантська китайська "white-label" IoT-платформа. Переважна більшість (за деякими оцінками, понад 80%) доступних на ринку "розумних" Wi-Fi розеток, ламп, реле та датчиків від сотень різних брендів насправді працюють на хмарній інфраструктурі Tuya (Рисунок1.6). Це де-факто стандарт масового ринку завдяки своїй дешевизні та простоті, але саме він несе в собі найбільші архітектурні недоліки.

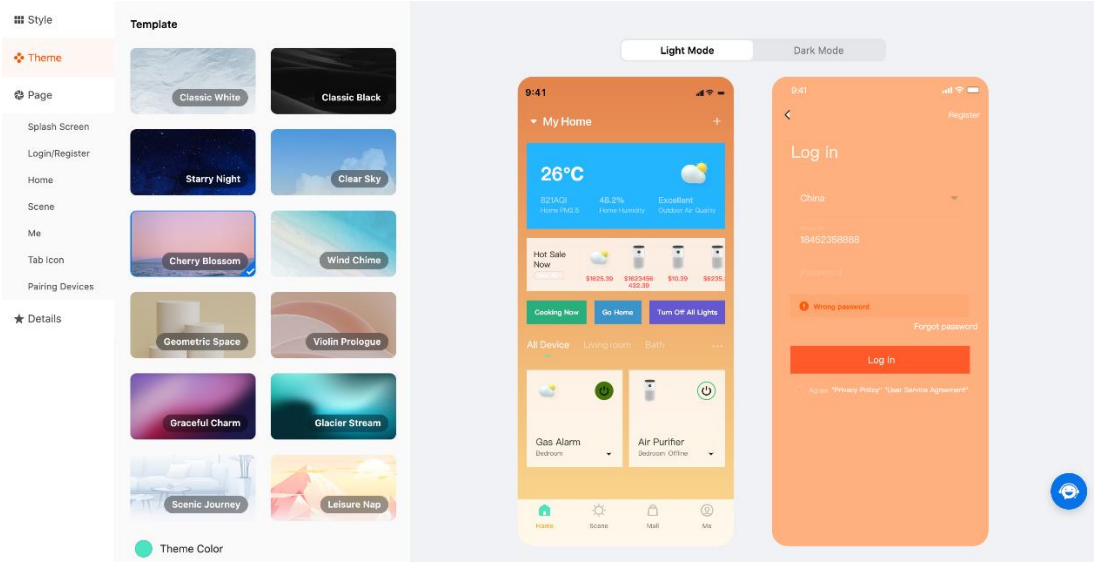


Рисунок 1.6 Інтерфейс програми Tuya

- **DIY-пристрої (Shelly, Sonoff):** Ці бренди орієнтовані на ентузіастів та професійних інсталяторів. Вони пропонують більш гнучкі пристрої (наприклад, реле Shelly), які часто дозволяють встановлення альтернативних прошивок (Tasmota, ESPHome) та роботу в локальних мережах, що частково знімає деякі обмеження.

1.2.3 Ключові недоліки масових IoT-рішень в контексті України

При детальному аналізі виявляється, що жодне з перерахованих масових рішень не здатне ефективно вирішити поставлену в даній роботі задачу (оптимізація енергоспоживання в кризових умовах) через низку фундаментальних недоліків.

1. Архітектурний недолік: Критична залежність від хмарних сервісів (Cloud Dependency) Це найвразливіше місце більшості масових систем, особливо платформи Tuya. Логіка роботи, сценарії автоматизації та навіть просте керування пристроєм через додаток виконуються **не локально**, а через віддалений сервер, що фізично знаходиться в Китаї, Європі чи США.

- **Проблема для України:** В умовах цілеспрямованих атак агресора на енергетичну інфраструктуру, відключення електроенергії неминуче призводить до зупинки роботи роутерів та обладнання інтернет-провайдерів. У момент зникнення інтернет-зв'язку вся "розумна" система, що залежить від хмари, миттєво стає **недієздатною**. "Розумний" термостат перетворюється на звичайний вимикач, а всі сценарії економії припиняють роботу. Така архітектурна вразливість є **неприйнятною** для критичних систем життєзабезпечення, якою є опалення.

2. Функціональний недолік: Спрощена та реактивна логіка керування Логіка 99% масових систем базується на простих сценаріях "Якщо-тоді" (IFTTT - If This, Then That). Наприклад: *«ЯКЩО час 18:00, ТО увімкнути світло»* або *«ЯКЩО температура в кімнаті < 20°C, ТО увімкнути обігрівач»*.

- **Проблема:** Це не **оптимізація**, а **автоматизація**. Система не приймає інтелектуальних *рішень*, а лише сліпо *виконує* жорсткий сценарій. Такий підхід є **реактивним** – система реагує на подію, що *вже відбулася* (стало холодно). Вона не є **проактивною** – не здатна спрогнозувати ситуацію (наприклад: "Температура 20.5°C, *але* це сонячний бік, *і* прогноз погоди обіцяє потепління за годину, *тому* вмикати опалення зараз недоцільно").

3. Недолік інтеграції: Закриті екосистеми ("Walled Gardens") Як зазначалося, провідні екосистеми є "садами за стінами". Виробники не надають відкритого API (програмного інтерфейсу) для глибокої інтеграції.

- **Проблема:** Користувач не може, наприклад, взяти дані з термостата Google Nest і використати їх у **власній, кастомній моделі керування**. Він змушений користуватися лише тими інструментами, які надав йому виробник. Це унеможливлює реалізацію власної наукової моделі, адаптованої під специфічні потреби.

4. Контекстний недолік: Ігнорування критичних зовнішніх даних Масові системи орієнтовані на стабільні умови експлуатації. Вони фокусуються на *внутрішніх* даних (температура в кімнаті, чи вдома користувач).

- **Проблема для України:** Вони не мають жодних механізмів для отримання та обробки *зовнішніх* даних, які є критично важливими саме для українських реалій:
    - **Графіки стабілізаційних відключень** (наприклад, від ДТЕК, Yasno чи інших обленерго).
    - **Прогноз погоди** (для проактивного нагріву/охолодження будинку до настання пікових температур).
    - **Поточне навантаження на енергосистему** (сигнали "Укренерго" про години пікового дефіциту). "Розумний" термостат Tuya з таким самим успіхом увімкне потужний електричний котел о 19:00 (у пік споживання), як і о 03:00 ночі, чим лише погіршує дефіцит в енергосистемі.

5. Цільовий недолік: Пріоритет комфорту над оптимізацією Маркетинг цих продуктів продає "зручність" (наприклад, "увімкніть чайник голосом"), а не "ефективність" (наприклад, "ми зекономили вам 30% на рахунку за опалення"). Їхні базові алгоритми не розроблені для вирішення математичної задачі оптимізації – **мінімізації цільової функції** (споживання кВт*год) при дотриманні **обмежень** (заданий рівень комфорту).

1.2.4 Висновки до підрозділу

Проведений аналіз ринку масових IoT-рішень для "розумного будинку" показує, що, незважаючи на їхню доступність та популярність, вони є переважно інструментами для **комфорту та простої автоматизації**, а не для **ефективної оптимізації енергоспоживання**.

Ключові недоліки – критична залежність від хмарних серверів (що неприпустимо в умовах блекаутів), закритість екосистем та використання спрощеної реактивної логіки, яка не враховує специфічних українських реалій, – роблять їх непридатними для вирішення поставленої задачі.

Таким чином, виникає чітко виражений **технологічний розрив (The Gap)**:

1. З одного боку, існують потужні, надійні та інтелектуальні **професійні BEMS** (розглянуті в 1.1), але вони є занадто дорогими та складними для масового сектору.
2. З іншого боку, існують доступні та зручні **масові IoT-рішення** (розглянуті в 1.2), але вони є ненадійними ("хмарозалежність") та функціонально обмеженими ("реактивна логіка").

Цей розрив змушує шукати третій шлях: використання **відкритих апаратних та програмних платформ**, які дозволяють поєднати доступність та гнучкість IoT з надійністю (локальне керування) та інтелектом (впровадження власної моделі) професійних BEMS.

1.3 Огляд відкритих апаратних та програмних платформ для побудови гнучких IoT-систем

1.3.1 Необхідність відкритих платформ та парадигма "Local-First"

Виявлені в підрозділі 1.2 фундаментальні недоліки масових IoT-рішень — критична залежність від хмарних серверів, закритість екосистем та спрощена логіка — роблять їх непридатними для побудови надійної та ефективної системи енергоменеджменту, особливо в кризових українських реаліях. Відмова системи опалення через відсутність інтернет-зв'язку є неприйнятним ризиком.

Це змушує шукати альтернативну архітектуру, що базується на принципово інших засадах. Такою альтернативою є парадигма **"Local-First" (локальне керування)**. Суть цієї парадигми полягає в тому, що всі критично важливі функції системи (збір даних, виконання логіки, прийняття рішень) повинні виконуватися **в межах локальної мережі будинку** (LAN/Wi-Fi) і не залежати від наявності підключення до Інтернету. Інтернет-з'єднання в такій архітектурі використовується лише для некритичних завдань: віддаленого доступу (за бажанням), завантаження прогнозів погоди чи оновлень.

Для реалізації архітектури "Local-First" необхідний набір відкритих, гнучких та сумісних між собою апаратних та програмних "будівельних блоків".

1.3.2 Апаратні платформи (Компонентна база)

Відкриті платформи дозволяють вільно обирати та комбінувати компоненти для створення кастомізованої системи. Їх можна розділити на два основні рівні.

1.3.2.1 Рівень периферійних пристроїв (Сенсори та Виконавчі механізми)

Це "очі" та "руки" системи. На ринку DIY (Do It Yourself) та гнучких IoT-рішень домінують дві основні технології:

- **Мікроконтролери з Wi-Fi (ESP32, ESP8266):** Продукти компанії Espressif Systems стали де-факто стандартом для розробки IoT-пристроїв. Їхні ключові переваги:
  - **Низька вартість:** Вартість плати розробника є вкрай низькою.
  - **Вбудований зв'язок:** Наявність модулів Wi-Fi та/або Bluetooth "з коробки".
  - **Достатня потужність:** 32-бітні процесори здатні не лише збирати дані з десятків сенсорів (температури, вологості, тиску, CO2, освітленості), але й керувати реле та навіть виконувати просту логіку.
  - **Величезна спільнота:** Наявність мільйонів прикладів коду, бібліотек та готових проектів.
- **Енергоефективні протоколи (Zigbee / Z-Wave):** Це технології бездротового зв'язку, спеціально розроблені для "розумного будинку". Їхня головна відмінність від Wi-Fi:
  - **Низьке енергоспоживання:** Сенсори, що працюють за протоколом Zigbee, можуть функціонувати роками від однієї батарейки (типу "таблетка"), що неможливо для Wi-Fi.
  - **Mesh-мережі (комірчасті мережі):** Кожен пристрій, що має постійне живлення (напр., "розумна" лампа чи реле), автоматично виступає ретранслятором (роутером) сигналу для інших пристроїв. Це створює надійну, самовідновлювану мережу, яка покриває великі площі та "обходить" перешкоди, що значно надійніше, ніж централізована Wi-Fi мережа, залежна від одного роутера.
  - *Недолік:* Для їхньої роботи потрібен окремий апаратний **координатор** (зазвичай у вигляді USB-стіка), який підключається до центрального шлюзу.

1.3.2.2 Рівень локального сервера / шлюзу (Gateway / Hub)

Це "мозок" локальної системи, який повністю замінює собою хмарні сервери. На відміну від мікроконтролерів, які лише виконують мікропрограми, шлюз є повноцінним комп'ютером.

- **Одноплатні комп'ютери (SBC - Single-Board Computers):** Найпопулярнішим представником цього класу є **Raspberry Pi** [17]. Його роль в архітектурі "Local-First" є ключовою:
  - **Операційна система:** Він працює під керуванням повноцінної ОС (зазвичай Linux), що дозволяє запускати складне серверне програмне забезпечення.

- **Достатні ресурси:** На відміну від ESP32, Raspberry Pi має потужний процесор та значний об'єм оперативної пам'яті. Це дозволяє йому одночасно виконувати функції:

1. Сервера автоматизації (п. 1.3.3.2).
2. MQTT-брокера (п. 1.3.3.1).
3. Бази даних (напр., InfluxDB) для зберігання історії показників.
4. **Виконавця складної математичної моделі** (наприклад, написаної на Python), що є ядром даної дипломної роботи.

- **Низьке енергоспоживання:** Споживаючи всього кілька ват, Raspberry Pi може працювати 24/7 від джерела безперебійного живлення (ДБЖ), забезпечуючи повну працездатність системи навіть під час відключень електроенергії (за умови живлення сенсорів та виконавчих пристроїв).
-

1.3.3 Програмні платформи (Програмна "зв'язка")

Самі по собі апаратні "цеглинки" не є системою. Їх об'єднує відкрите програмне забезпечення, що забезпечує зв'язок та логіку.

1.3.3.1 Протокол зв'язку: MQTT

MQTT (Message Queuing Telemetry Transport) — це стандартний, легковагий протокол для IoT-телеметрії [13]. Він є "нервовою системою" локальної мережі та працює за принципом **"Видавець-Підписник" (Publish/Subscribe)**, на відміну від класичного "Запит-Відповідь" (Request-Response).

- **Архітектура:** У центрі системи (на Raspberry Pi) працює **MQTT-брокер** (напр., Mosquitto) – це "поштове відділення" для всіх повідомлень.
- **Видавець (Publisher):** Сенсор температури (ESP32) не надсилає дані напряму контролеру. Він "публікує" своє повідомлення (напр., 21.5) у певну "тему" (напр., home/living_room/temperature) на брокер.
- **Підписник (Subscriber):** Наша модель (на тому ж Raspberry Pi) "підписана" на цю тему. Як тільки брокер отримує нове повідомлення в цій темі, він миттєво доставляє його всім підписникам.
- **Ключові переваги:**

1. **Асинхронність:** Компонентам не потрібно чекати один на одного.
2. **Легковаговість:** Протокол вимагає мінімального трафіку.
3. **Роз'єднаність (Decoupling):** Сенсор (видавець) не знає нічого про існування моделі (підписника), і навпаки. Це дозволяє додавати або замінювати компоненти "на льоту", не зупиняючи систему.

1.3.3.2 Платформи локальної автоматизації

Це ПЗ, що запускається на Raspberry Pi, збирає дані з MQTT та виконує логіку — від простих сценаріїв до нашої складної математичної моделі.

- **Home Assistant (HA) [21]:** На сьогодні це найпотужніша open-source платформа для "розумного будинку". Її переваги:
  - **"Local-First" в основі:** Розроблена для роботи без Інтернету.
  - **Величезна кількість інтеграцій:** Підтримує тисячі пристроїв різних виробників "з коробки".
  - **Потужний двигун автоматизації:** Дозволяє створювати складні сценарії.
  - **Інтерфейс:** Надає готовий мобільний та веб-інтерфейс для керування.
- **Node-RED:** Це візуальний інструмент для програмування потоків даних (flow-based programming). Замість написання коду, користувач з'єднує логічні блоки (ноди). Ідеально підходить для прототипування складної логіки та нестандартних інтеграцій, наприклад, парсингу (зчитування) даних про графіки відключень з веб-сайту ДТЕК та передачі їх у модель.

1.3.3.3 Прошивки для периферійних пристроїв

Щоб змусити ESP32 працювати локально через MQTT, потрібне спеціальне ПЗ (прошивка), яке замінює собою пропрієтарні хмарні прошивки (як у Tuya).

- **ESPHome / Tasmota:** Це відкриті прошивки, які є **антиподом** до хмарних рішень. Вони розроблені для глибокої інтеграції з Home Assistant та MQTT "з коробки" і гарантують **нульову залежність від хмарних серверів**.

1.3.4 Висновки до підрозділу

Аналіз показав, що на ринку існує повний набір відкритих, доступних, надійних та гнучких апаратних і програмних "будівельних блоків".

Комбінація цих інструментів (наприклад, архітектура: Сенсори на ESP32/Zigbee + прошивка ESPHome + Raspberry Pi + MQTT-брокер Mosquitto + платформа Home Assistant) дозволяє повністю вирішити всі фундаментальні проблеми масових рішень, окреслені в підрозділі 1.2:

1. **Проблема хмарної залежності** вирішується парадигмою "Local-First" (використанням Raspberry Pi як локального сервера).
2. **Проблема закритих екосистем** вирішується використанням відкритих протоколів (MQTT) та платформ (Home Assistant), які здатні інтегрувати пристрої різних виробників.

3. **Проблема реактивної логіки** вирішується можливістю запуску на Raspberry Pi власної, кастомної логіки.

   Таким чином, ми визначили надійний та гнучкий **інструментарій** для побудови системи. Тепер відкритим залишається головне наукове питання, яке ми дослідимо в наступному підрозділі: яку саме **інтелектуальну логіку (математичну модель)** ми будемо реалізовувати за допомогою цих інструментів, щоб досягти мети енергоефективності.

1.4. Аналіз існуючих математичних моделей керування енергоспоживанням

1.4.1 Від інструментів до інтелекту

Попередній підрозділ (1.3) продемонстрував, що на ринку існує повний набір доступних, відкритих та надійних апаратних (Raspberry Pi, ESP32, Zigbee) та програмних (MQTT, Home Assistant, ESPHome) "будівельних блоків". Цей інструментарій дозволяє створити відмовостійку "Local-First" архітектуру, яка вирішує проблеми хмарної залежності та закритих екосистем.

Однак, наявність інструментів не вирішує головного завдання. Система керування є настільки ефективною, наскільки інтелектуальною є її **логіка прийняття рішень (модель)**.

Мета даного підрозділу — провести системний аналіз та порівняння основних класів математичних моделей, що застосовуються в BEMS та системах "розумного будинку", для наукового обґрунтування вибору методології, яка буде покладена в основу даної дипломної роботи.

1.4.2 Модель 1: Керування на основі жорстких правил та розкладів (Rule-Based / Schedule-Based Control)

Це найбільш базовий та історично перший підхід до автоматизації. Він базується на детермінованій, бінарній логіці та жорстко заданих часових графіках.

- **Суть методу:** Система працює за набором статичних правил "ЯКЩО-ТОДІ" (IF-THEN) та часових уставок, які задаються користувачем.
- **Приклади:**
  - *Термостат:* ЯКЩО Temperature_Sensor < 20.0 ТО Увімкнути_Котел.
  - *Розклад:* ЯКЩО Time > 23:00 ТО Встановити_Уставку_18.0.
  - *Присутність:* ЯКЩО Motion_Sensor = FALSE протягом 30_хвилин ТО Вимкнути_Опалення_в_Зоні.

- **Переваги:**
  - **Простота:** Легкість в реалізації, налаштуванні та налагодженні.
  - **Низькі обчислювальні вимоги:** Може виконуватись на найпростіших мікроконтролерах.
  - **Передбачуваність:** Поведінка системи є на 100% детермінованою та зрозумілою.
- **Недоліки (Критика):**
  - **Повна відсутність адаптивності:** Система є "сліпою" до будь-яких умов, що не прописані в правилах. Вона не враховує теплову інерцію будівлі, прогноз погоди, сонячну радіацію чи зміну звичок мешканців (наприклад, незапланований робочий день з дому).
  - **Низька ефективність:** Призводить до частих "перельотів" (over/undershoot) температури та неефективного використання енергії.
- **Висновок:** Цей підхід є *автоматизацією*, але не *оптимізацією*. Він не відповідає вимогам до сучасної ефективної системи і не несе в собі наукової новизни[2].
- **,ele**

1.4.3 Модель 2: Статистичні та регресійні моделі (Statistical Models)

Цей підхід є кроком вперед, оскільки він намагається описати поведінку системи на основі аналізу історичних даних, а не жорстких правил.

- **Суть методу:** Використання методів математичної статистики (найчастіше – множинної лінійної регресії) для побудови моделі, що пов'язує споживання енергії (або інший цільовий параметр) з набором вхідних факторів.
- Приклад: Будується модель прогнозування споживання енергії E у вигляді:

$$E = \beta_0 + \beta_1 \cdot T_{outside} + \beta_2 \cdot TimeOfDay + \beta_3 \cdot IsWeekend + \epsilon$$

де T_outside – зовнішня температура, TimeOfDay – час доби, IsWeekend – бінарний фактор, \beta_i – коефіцієнти, розраховані за історичними даними.

- **Переваги:**
  - Базується на реальних даних з конкретного об'єкта.
  - Більш гнучка, ніж статичні правила, і дозволяє робити прості прогнози.
- **Недоліки (Критика):**
  - **Низька точність:** Теплові процеси в будівлі (інерція, сонячна радіація, поведінка людей) є глибоко **нелінійними**. Лінійні моделі дають дуже грубе наближення і велику похибку.
  - **Складність врахування нечислових факторів:** Такі моделі погано справляються з нечіткими, "людськими" поняттями, як-от "комфорт" або "незначне потепління".

1.4.4 Модель 3: Моделі на основі штучного інтелекту (AI / Machine Learning)

Це найбільш сучасний та потужний клас моделей, що використовується в передових і-BEMS (інтелектуальних BEMS) та "розумних" термостатах (напр., Nest). Вони здатні "навчатися" на великих масивах даних.

- **Підкатегорія: Нейронні мережі (NN) та Модельне Предиктивне Керування (MPC):**
  - **Суть методу:** Це комбінований підхід. Спочатку **нейронна мережа** (часто рекурентна, RNN, або LSTM) навчається на величезному масиві історичних даних (погода, температура, споживання, дії користувача). Її завдання – створити "цифрового двійника" (Digital Twin) будівлі, тобто здатність **прогнозувати** стан системи (напр., *«якою буде температура в будинку через 2 години, якщо нічого не робити»*).
  - Далі, **MPC (Model Predictive Control)** використовує цю прогнозну модель. Кожні кілька хвилин MPC "програє" десятки сценаріїв на майбутнє (напр., "увімкнути котел на 10 хв", "увімкнути на 20 хв", "не вмикати") і обирає той, який мінімізує цільову функцію (напр., найменше споживання енергії при утриманні температури в "коридорі комфорту" протягом наступних 4 годин).
- **Переваги:**
  - **Найвища точність:** Здатні враховувати складні, приховані та нелінійні патерни (теплова інерція, звички мешканців).
  - **Проактивність:** Це по-справжньому "передбачувальне" керування.
- **Недоліки (Критика):**
  - **Проблема "Чорної скриньки" (Black Box):** Це головний недолік для наукової та інженерної роботи. Нейронна мережа не дає відповіді, *чому* вона прийняла те чи інше рішення. Її логіка непрозора [3]. Це ускладнює довіру до системи та її верифікацію.
  - **Потреба у величезних даних (Data-Hungry):** Для навчання якісної прогнозної моделі потрібні чисті, марковані дані за тривалий період (часто місяці або й роки) з конкретного об'єкта. Отримати такий набір даних в рамках дипломної роботи вкрай складно.
  - **Обчислювальна складність:** "Навчання" таких моделей вимагає значних ресурсів (часто GPU в хмарі). Виконання (inference) MPC-алгоритму в реальному часі також може бути надто важким завданням для малопотужного локального сервера типу Raspberry Pi.
  - **Складність модифікації:** Якщо у нас з'являється новий вхідний фактор (напр., графік відключень ДТЕК), ми не можемо просто "додати правило". Потрібна повна перебудова моделі та її перенавчання на нових даних, що інженерно дуже складно.

1.4.5 Модель 4: Моделі на основі нечіткої логіки (Fuzzy Logic Control, FLC)

Цей підхід, запропонований Лотфі Заде **[6]**, є потужною альтернативою як класичній бінарній логіці, так і складним нейронним мережам. Він базується на імітації людського мислення, яке оперує неточними, нечіткими поняттями.

- **Суть методу:** Нечітка логіка (FLC) працює не з бінарними значеннями (0 - 'холодно', 1 - 'тепло'), а з **ступенями належності (Membership Functions)** до нечітких множин.
- **Приклад:** Температура 21°C для FLC-контролера не є просто "не холодно". Вона може бути 'комфортною' на 90% і одночасно 'прохолодною' на 10%.
- **Принцип роботи:** FLC-система складається з трьох блоків:

1. **Фазифікація:** "Чіткі" вхідні дані (напр., T = 19.5°C) перетворюються на нечіткі (напр., 'Холодно' на 20%, 'Прохолодно' на 80%).
2. **База правил (Rule Base):** Це "мозок" системи, складений "експертом" (інженером, тобто нами). Правила пишуться лінгвістично, імітуючи логіку людини.
3. **Дефазифікація:** Результати виконання правил (напр., "опалення має бути Трохи' і 'Середньо') знову перетворюються на чіткий керуючий сигнал (напр., "встановити потужність котла на 35%").

- **Приклад правил для нашої задачі:**
  - ЯКЩО Температура 'Холодна' ТО Опалення 'Сильне'
  - ЯКЩО Температура 'Прохолодна' І ПрогнозПогоди 'Сонячний' ТО Опалення 'Слабке' (проактивне врахування майбутнього тепла від сонця)
  - ЯКЩО Температура 'Комфортна' І ГрафікВідключень 'Скоро' ТО Опалення 'Середнє' (проактивний нагрів будинку *до* блекауту)
- **Переваги (Ключове обґрунтування для даної роботи):**

1. **Робота з невизначеністю:** FLC ідеально підходить для опису неточних, "людських" понять: 'Комфортно', 'Трохи холодно', 'Скоро', 'Дуже сонячно'.
2. **Інтерпретованість ("Біла скринька"):** На відміну від нейромереж, логіка FLC є повністю прозорою. База правил легко читається, аналізується та верифікується[4].
3. **Гнучкість та легкість модифікації:** Це **критична перевага** для нашої задачі. Ми можемо легко додати новий вхідний параметр (напр., "Ціна_Енергії" або "Стан_Відключень"), просто доповнивши базу правил новими лінгвістичними конструкціями, не руйнуючи всю модель.
4. **Низькі обчислювальні вимоги:** FLC-модель **не потребує навчання** на величезних масивах даних. Вона базується на експертних знаннях. Процес обчислення (inference) є швидким математичним розрахунком (операції з матрицями), який легко виконується в реальному часі навіть на Raspberry Pi.

- **Недоліки:** Ефективність моделі прямо залежить від якості складених "експертом" правил та налаштування функцій належності.

1.4.6 Порівняльний аналіз та висновки до підрозділу

Проведений аналіз дозволяє звести характеристики розглянутих моделей у порівняльну таблицю (Таблиця 1.1).

Таблиця 1.1 — Порівняльний аналіз математичних моделей керування

Критерій	**Модель 1: Правила/Розклад**	**Модель 2: Статистика**	**Модель 3: Нейромережі (NN/MPC)**	**Модель 4: Нечітка логіка (FLC)**
Адаптивність	Відсутня	Низька (лише до даних, що були в минулому)	**Висока** (навчається)	**Висока** (реакція на нечіткі входи)
Врахування нелінійності	Низьке	Низьке	**Висока**	**Висока**
Робота з неточними/людськими поняттями	Відсутня	Відсутня	Низька	**Висока**
Інтерпретованість ("Біла скринька")	Висока	Середня	**Дуже низька ("Чорна скринька")**	**Висока**
Потреба у даних для навчання	Низька	Середня	**Дуже висока**	**Низька** (потребує експертних знань)

Критерій	Модель 1: Правила/Розклад	Модель 2: Статистика	Модель 3: Нейромережі (NN/MPC)	Модель 4: Нечітка логіка (FLC)
Обчислювальна складність (Inference)	Дуже низька	Низька	Висока	Низька/Середня
Простота модифікації (додання нових факторів)	Середня	Складна	**Дуже складна**	**Висока**

Висновки:

Аналіз та порівняння показують, що:

1. **Моделі 1 (Правила) та 2 (Статистика)** є занадто простими, неадаптивними та нездатними вирішити складну задачу оптимізації в динамічному середовищі.
2. **Модель 3 (Нейромережі)**, хоч і є найпотужнішою з точки зору прогнозування, має критичні недоліки для даної роботи: вона є "чорною скринькою", вимагає величезних обсягів даних для навчання та вкрай складна в модифікації для врахування специфічних українських реалій (напр., ручне додавання фактору графіків відключень).
3. **Модель 4 (Нечітка логіка)** є **оптимальним інженерним та науковим компромісом**. Вона поєднує *адаптивність* та здатність *працювати зі складними, нелінійними та неточними вхідними даними* (як AI), але при цьому залишається *прозорою ("біла скринька"), легкою для модифікації* та *обчислювально простою*.

Таким чином, проведений аналіз обґрунтовує вибір **теорії нечітких множин** як основного математичного апарату для розробки шуканої моделі керування енергоефективністю. Наступний підрозділ формулює фінальні завдання дослідження, виходячи з цього вибору.

Висновки до Розділу 1

У першому розділі було проведено глибокий системний аналіз та огляд літератури з проблеми керування енергоефективністю будівель. Аналіз дозволив зробити наступні ключові висновки:

1. **Виявлено технологічний розрив:** Існує значний розрив між двома основними сегментами ринку. З одного боку – професійні системи BEMS, які є потужними, але надзвичайно дорогими, складними та орієнтованими на великі комерційні об'єкти. З іншого – масові IoT-рішення ("розумний будинок"), які є доступними, але мають фундаментальні недоліки, що роблять їх непридатними для вирішення задачі енергетичної стійкості в умовах України.
2. **Ідентифіковано критичні недоліки масових рішень:** Доведено, що головними проблемами масових платформ (таких як Tuya, Google Home) є:
   - **Хмарна залежність (Cloud Dependency):** Непрацездатність системи за відсутності Інтернету, що є неприйнятним для критичної інфраструктури (опалення) в умовах блекаутів.
   - **Закритість екосистем:** Неможливість гнучкої модифікації та інтеграції власних алгоритмів.
   - **Спрощена логіка:** Використання реактивних сценаріїв "Якщо-Тоді" замість проактивної оптимізації.
3. **Обґрунтовано вибір апаратно-програмного базису:** Встановлено, що недоліки масових систем можуть бути повністю усунені шляхом застосування відкритих апаратних (Raspberry Pi, ESP32) та програмних (MQTT, Home Assistant) платформ. Така комбінація дозволяє реалізувати надійну, гнучку та відмовостійку архітектуру за принципом **"Local-First" (локальне керування)**.
4. **Обґрунтовано вибір математичного апарату:** Проведено порівняльний аналіз основних класів математичних моделей керування (на основі правил, статистики, нейронних мереж та нечіткої логіки). Доведено, що **теорія нечітких множин (Fuzzy Logic Control)** є **оптимальним вибором** для поставленої задачі. На відміну від "чорних скриньок" нейронних мереж, FLC є прозорою ("білою скринькою"), обчислювально ефективною, не вимагає величезних масивів даних для навчання та ідеально підходить для роботи з неточними, "людськими" поняттями (напр., 'комфортно', 'скоро') та легкої модифікації (напр., додавання фактору графіків відключень).

Таким чином, проведений у Розділі 1 аналіз повністю підтверджує актуальність теми, дозволяє чітко сформулювати науково-технічне протиріччя та обґрунтувати вибір інструментарію (Local-First IoT) та математичного апарату (FLC) для його вирішення.

1.5. Постановка завдань дослідження

На основі висновків, зроблених у даному розділі, для досягнення поставленої у вступі мети (підвищення енергоефективності будинку шляхом розробки адаптивної моделі керування) необхідно вирішити наступні **науково-практичні завдання**:

1. **Розробити математичну модель** адаптивного керування енергоефективністю на основі теорії нечіткої логіки. Це включає формалізацію вхідних (сенсори, погода, графіки відключень) та вихідних (керуючі дії) змінних, розробку функцій належності та формування експертної бази правил.
2. **Спроектувати архітектуру** інформаційної системи, що реалізує розроблену модель. Архітектура має базуватися на відкритих платформах (Raspberry Pi, ESP32) та протоколі MQTT, забезпечуючи надійну роботу за принципом "Local-First".
3. **Розробити програмне забезпечення** для ключових вузлів системи: програмний комплекс для центрального шлюзу (реалізація FLC-моделі) та мікропрограми для периферійних сенсорних та виконавчих вузлів.
4. **Провести експериментальне дослідження** ефективності розробленої моделі шляхом імітаційного комп'ютерного моделювання. Виконати порівняльний аналіз показників енергоспоживання та підтримки комфорту для запропонованої моделі та базових сценаріїв керування (простий термостат, керування за розкла

РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ АДАПТИВНОГО КЕРУВАННЯ ЕНЕРГОЕФЕКТИВНІСТЮ

2.1 Обґрунтування вибору математичного апарату

Вирішення задачі керування енергоефективністю будинку є складною, багатофакторною проблемою. Об'єкт керування – житлова будівля – характеризується значною тепловою інерцією, нелінійними динамічними процесами та високим ступенем невизначеності вхідних даних. Ці невизначеності пов'язані як з зовнішніми факторами (стохастичний характер погоди), так і з внутрішніми (непередбачувана поведінка та суб'єктивне відчуття "комфорту" мешканців).

Проведений у Розділі 1.4 аналіз існуючих математичних моделей керування показав їхні суттєві обмеження стосовно поставленої задачі:

1. **Моделі на основі жорстких правил та розкладів** є детермінованими та неадаптивними. Вони не здатні реагувати на динамічні зміни в середовищі та не враховують нечіткий характер людського поняття "комфорту".
2. **Статистичні (регресійні) моделі** погано описують складні нелінійні теплові процеси в будівлі та вимагають чіткої формалізації всіх вхідних факторів, що складно для поведінкових патернів.
3. **Моделі на основі нейронних мереж (NN)**, хоча й демонструють високу точність у прогнозуванні на основі великих даних, мають два критичних недоліки для даної роботи:
    - **Проблема "чорної скриньки"**: Логіка прийняття рішень нейромережею є непрозорою, що ускладнює верифікацію, налагодження та сертифікацію моделі для керування критичною інфраструктурою.
    - **Складність модифікації**: Для додавання нового критичного фактору (наприклад, раптової появи графіків стабілізаційних відключень) потрібне повне перенавчання моделі на нових масивах даних, що є інженерно складним та негнучким рішенням.

На противагу цим методам, **теорія нечітких множин та нечіткої логіки (Fuzzy Logic) [6]** надає математичний апарат, що ідеально відповідає природі поставленої задачі.

Вибір теорії нечіткої логіки як базису для розробки моделі керування обґрунтовується такими її перевагами:

1. **Здатність до формалізації нечітких та якісних понять.** Ключовий параметр, який ми оптимізуємо, – "комфорт" – не є чіткою фізичною

величиною. Це суб'єктивне поняття, яке людина описує лінгвістичними термінами: 'прохолодно', 'комфортно', 'тепло'. Нечітка логіка дозволяє безпосередньо оперувати цими неточними, лінгвістичними змінними.
2. **Робота з нелінійними системами.** FLC-контролери є універсальними апроксиматорами, здатними ефективно описувати та керувати складними нелінійними динамічними системами, якою і є теплова модель будинку.
3. **Прозорість та інтерпретованість ("Біла скринька").** На відміну від нейронних мереж, "мозок" нечіткої моделі – це **база правил (Rule Base)**, складена експертом (інженером). Правила мають інтуїтивно зрозумілий вигляд: ЯКЩО Температура 'Низька' І Прогноз 'Сонячний', ТО Опалення 'Середнє'. Цю логіку легко читати, аналізувати, верифікувати та налагоджувати.
4. **Гнучкість та легкість модифікації.** Це критична перевага для умов України. Якщо виникає новий вхідний фактор (наприклад, "Наявність_електроенергії" зі станами 'Є', 'Скоро_відключення', 'Блекаут'), не потрібно перенавчати всю модель. Достатньо додати нову вхідну лінгвістичну змінну та розширити базу правил, що є значно простішим інженерним завданням.
5. **Обчислювальна ефективність.** Алгоритми нечіткого висновку (Mamdani, Sugeno) не вимагають "навчання" на великих масивах даних і зводяться до швидких математичних операцій (робота з матрицями, пошук центру ваги) [8], що легко реалізуються в реальному часі на малопотужних обчислювальних пристроях, таких як Raspberry Pi.

Таким чином, враховуючи нелінійний характер об'єкта керування, необхідність роботи з нечіткими поняттями ("комфорт") та жорсткі вимоги до гнучкості моделі (адаптація до графіків відключень), **теорія нечіткої логіки** обирається як основний математичний апарат для розробки адаптивної моделі керування енергоефективністю в даній кваліфікаційній роботі.

Гаразд, переходимо до формалізації нашої задачі. Це один з найважливіших етапів, де ми переводимо інженерну ідею в чітку мову математики.

2.2 Математична постановка задачі оптимізації

Розробка моделі керування енергоефективністю є класичною задачею оптимізації. Нам необхідно знайти такий закон керування U, який мінімізує цільову функцію (споживання енергії) при дотриманні низки заданих обмежень (комфорт мешканців).

2.2.1 Визначення цільової функції та обмежень системи

Цільова функція

Метою системи є мінімізація сумарного споживання енергії E виконавчими пристроями (в даній роботі – системою опалення та системою освітлення) за певний проміжок часу T (наприклад, 24 години).

Цільову функцію J можна представити у вигляді:

$$J = \int_0^T (P_{heat}(t) + P_{light}(t))\, dt \to \min$$

де:

- T – часовий горизонт оптимізації.
- P_heat (t) – миттєва потужність, що споживається системою опалення в момент часу (t).
- P_light (t) – миттєва потужність, що споживається системою освітлення в момент часу (t).

Керування системою опалення є головним пріоритетом, оскільки його частка в загальному енергоспоживанні будинку є домінуючою (часто понад 70-80%).

Обмеження системи

Мінімізація функції J не може бути абсолютною (тобто просто вимкнути все), вона має виконуватися при дотриманні головного обмеження – підтримання комфортних умов для мешканців.

Обмеження можна формалізувати наступним чином:

1. Обмеження по комфортній температурі: Внутрішня температура T_in (t) повинна знаходитись у заданому комфортному діапазоні: [T_min_comfort, T_max_comfort] протягом часу, коли мешканці активні.

$$T_{in}(t) \in [T_{min\_comfort}, T_{max\_comfort}], \quad \forall t \in T_{presence}$$

При цьому, в "економному" режимі (вночі, або коли мешканці відсутні), діапазон може зміщуватися до [T_min_eco, T_max_eco].

2. Обмеження по освітленості: Рівень освітленості L_in (t) у житлових зонах має бути не нижчим за санітарні норми L_norm (напр., 150 лк) у вечірній час або при недостатньому природному освітленні.

$$P_{total}(t) \leq P_{backup}, \quad \forall t \in T_{blackout}$$

3. Зовнішні примусові обмеження (для умов України): Це специфічне обмеження, що має вищий пріоритет. Споживання потужності P_total (t) має дорівнювати нулю або бути мінімальним (робота від ДБЖ) під час графіків стабілізаційних відключень T_blackout.

$$P_{total}(t) \leq P_{backup}, \quad \forall t \in T_{blackout}$$

Таким чином, задача зводиться до **задачі умовної оптимізації** – мінімізації енергоспоживання при жорсткому дотриманні обмежень комфорту та енергетичної доступності.

2.2.2 Формалізація вхідних та вихідних змінних

Для вирішення цієї задачі наша модель (контролер нечіткої логіки) повинна в кожний момент часу t аналізувати вектор вхідних змінних X(t) та формувати на їх основі вектор вихідних (керуючих) змінних Y(t).

1. Вектор вхідних змінних X(t)

Вхідні змінні є вимірюваними або прогнозованими параметрами, що описують поточний стан системи та зовнішнього середовища. Їх можна розділити на кілька груп:

Внутрішні вимірювані параметри:

- x_1 = T_in(t): Поточна температура всередині приміщення (°C).
- x_2 = e_T(t): Похибка регулювання температури (e_T(t) = T_setpoint - T_in(t)).
- x_3 = \fracdT_indt: Швидкість зміни температури (градієнт), що показує, охолоджується чи нагрівається приміщення.
- x_4 = H_in(t): Поточна вологість всередині приміщення (%).
- x_5 = L_in(t): Поточний рівень освітленості в зоні (лк).
- x_6 = Presence(t): Факт присутності мешканців (бінарний або ймовірнісний).

Зовнішні вимірювані/прогнозовані параметри:

- x_7 = T_out(t): Поточна температура зовні (°C).
- x_8 = T_forecast(t+\Delta t): Прогнозована температура на \Delta t годин вперед.
- x_9 = Weather(t): Поточні/прогнозовані погодні умови (напр., 'Сонячно', 'Хмарно', 'Опади').
- x_10 = Time(t): Поточний час доби (для визначення режимів "день/ніч/пік").

Специфічні (контекстні) параметри для України:

- x_11 = Blackout(t+\Delta t): Статус відключень (напр., 'Є живлення', 'Відключення через годину', 'Відключено').

2. Вектор вихідних (керуючих) змінних Y(t)

Вихідні змінні – це чіткі команди, які наша модель має видати виконавчим пристроям.

- y_1 = P_heat_cmd(t): Керуючий сигнал для системи опалення (напр., від 0% до 100% потужності котла, або час увімкнення/вимкнення).
- y_2 = L_light_cmd(t): Керуючий сигнал для системи освітлення (напр., рівень димування від 0% до 100%).

Отже, задача моделювання полягає у знаходженні нелінійного відображення (функції) F:

$$Y(t) = F(X(t))$$

де функція F реалізована у вигляді системи нечіткого логічного висновку (Fuzzy Logic Inference System) та оптимізує цільову функцію J при дотриманні заданих обмежень. Розробці цієї функції F (фазифікації, базі правил та дефазифікації) присвячений наступний підрозділ.

2.3 Розробка моделі нечіткого висновку (Fuzzy Logic Inference System)

Як було обґрунтовано в 2.1, для реалізації нашої функції керування Y(t) = F(X(t)) ми використовуємо апарат нечіткої логіки. Система нечіткого логічного висновку (FLC) є "мозком" нашого контролера. Вона імітує процес прийняття рішень людиною-експертом, переводячи чіткі числові дані в якісні лінгвістичні оцінки, обробляючи їх за допомогою бази правил, і повертаючи чіткий керуючий сигнал.

Класична система нечіткого висновку (ми будемо використовувати найбільш поширений та інтуїтивно зрозумілий алгоритм Мамдані (Mamdani) **[7]**) складається з трьох послідовних етапів:

1. **Фазифікація (Fuzzification):** Перетворення чітких вхідних значень (напр., "температура = 19.5°C") у нечіткі (напр., "кімната 'Прохолодна' на 80%").
2. **Агрегація (Rule Evaluation):** Застосування бази експертних правил до нечітких значень для отримання нечіткого результату.
3. **Дефазифікація (Defuzzification):** Перетворення нечіткого результату (напр., "опалення має бути 'Середнім'") назад у чіткий керуючий сигнал (напр., "встановити потужність на 65%").

2.3.1 Фазифікація: Визначення лінгвістичних змінних та їх функцій належності

Перший і найважливіший крок – це **фазифікація**, або "розмиття" чітких вхідних даних. Для цього кожній вхідній та вихідній змінній (з визначених у 2.2.2) ми ставимо у відповідність **лінгвістичну змінну**.

Лінгвістична змінна – це змінна, значеннями якої є не числа, а слова або словосполучення природної мови (наприклад, 'Холодно', 'Тепло'). Кожне таке слово називається **термом (fuzzy set)**, і ступінь належності до нього описується **функцією належності (Membership Function, MF)**.

Функція належності \μ(x) показує ступінь (в діапазоні [0, 1]), з яким чітке значення x належить до нечіткого терму. Для нашої моделі ми будемо використовувати **трикутні (trimf)** та **трапецієподібні (trapmf)** функції належності, оскільки вони є обчислювально простими (що важливо для реалізації на Raspberry Pi) та легко піддаються інтуїтивному налаштуванню.

Для керування системою опалення (як найбільш пріоритетною) визначимо наступні ключові лінгвістичні змінні:

Вхідні Лінгвістичні Змінні (з вектора X(t))

1. LV1: "Похибка_Температури" (eT)

- **Опис:** Головний параметр керування. Показує різницю між *бажаною* температурою та *поточною*. e_T(t) = T_setpoint - T_in(t).
- **Універсум (Universe of Discourse):** [-5°C, +5°C].
- **Терми (Нечіткі множини):**
  - NV (Negative Big / Негативна Велика): Кімната сильно перегріта.
  - NS (Negative Small / Негативна Мала): Кімната трохи перегріта.
  - Z (Zero / Нульова): Температура відповідає бажаній.

- PS (Positive Small / Позитивна Мала): Кімнаті трохи прохолодно.
- PB (Positive Big / Позитивна Велика): Кімнаті дуже холодно.

- **Функції належності (\μ(eT)):** Трапецієподібні по краях (NV, PB) та трикутні в центрі (NS, Z, PS) для повного покриття універсуму.

2. LV2: "Градієнт_Температури" (deT)

- **Опис:** Похідна від похибки температури (\fracde_Tdt). Показує, *наскільки швидко* кімната охолоджується чи нагрівається. Це дозволяє системі реагувати на випередження.
- **Універсум:** [-1°C/год, +1°C/год].
- **Терми:**
  - N (Negative / Негативний): Кімната активно нагрівається (похибка зменшується).
  - Z (Zero / Нульовий): Температура стабільна.
  - P (Positive / Позитивний): Кімната активно охолоджується (похибка зростає).

3. LV3: "Прогноз_Погоди" (Forecast)

- **Опис:** Адаптивний параметр. Прогноз зовнішньої температури на найближчі \Delta t (напр., 3 години).
- **Універсум:** [-20°C, +30°C].
- **Терми:**
  - Мороз (Дуже холодно)
  - Холодно
  - Прохолодно
  - Тепло (Комфортна зовнішня температура)

4. LV4: "Статус_Відключень" (Blackout)

- **Опис:** Специфічний адаптивний параметр для умов України. Показує час до прогнозованого відключення електроенергії.
- **Універсум:** [0 годин, 6 годин]. (0 – відключено зараз).
- **Терми:**
  - Відключено (Живлення відсутнє)
  - Критично (Відключення протягом 1 години)
  - Скоро (Відключення протягом 1-3 годин)
  - Безпечно (Відключення не скоро, >3 годин)

Вихідна Лінгвістична Змінна (з вектора Y(t))

OV1: "Потужність_Опалення" (P_heat)

- **Опис:** Керуючий сигнал, що подається на виконавчий пристрій (напр., ШІМ-сигнал на твердотільне реле котла або сервопривід клапана).

- **Універсум:** [0%, 100%].
- **Терми:**
  - Вимкнено (0%)
  - Низька
  - Середня
  - Висока
  - Максимум (100%)

2.3.2 Розробка бази правил (Rule Base)

База правил (або база знань) є ядром системи нечіткого логічного висновку. Вона містить набір лінгвістичних правил у форматі **"ЯКЩО <*умова*>, ТО <*висновок*>"**, які кодують експертні знання та досвід інженера-теплотехніка або досвідченого користувача про те, як ефективно керувати системою опалення. Саме ця база правил реалізує нелінійне відображення Y(t) = F(X(t)), перетворюючи набір вхідних нечітких змінних у вихідну нечітку змінну.

Формат правил та логічні операції

Кожне правило складається з двох частин:

1. **Антецедент (antecedent):** Умовна частина правила, що починається з "ЯКЩО". Вона може містити одну або декілька умов, об'єднаних логічними операторами **І (AND)**, **АБО (OR)**.
2. **Консеквент (consequent):** Частина, що містить висновок і починається з "ТО". Вона визначає вихідну дію.

У нашій моделі ми будемо переважно використовувати оператор **І (AND)**, який в алгоритмі Мамдані зазвичай реалізується через операцію взяття **мінімуму** зі ступенів належності. Наприклад, якщо для правила ЯКЩО A І B ТО C ступінь належності до терму A дорівнює \μ_A=0.8, а до терму B – \μ_B=0.5, то ступінь істинності всього антецедента буде \min(0.8, 0.5) = 0.5.

Формування базової матриці правил (Fuzzy Associative Memory - FAM)

Для початку сформуємо основну логіку керування, що базується на двох найбільш критичних параметрах будь-якого ПІД-подібного регулятора: **похибці (eT)** та її **похідній (deT)**. Це дозволить системі не лише реагувати на поточну температуру, але й враховувати динаміку її зміни.

Базу правил для цих двох змінних зручно представити у вигляді матриці асоціативної пам'яті (Таблиця 2.1), де рядки відповідають термам змінної Похибка_Температури, стовпці – термам Градієнт_Температури, а комірки – результуючим термам вихідної змінної Потужність_Опалення.

Таблиця 2.1 — Базова матриця правил для керування опаленням

deT → eT ↓	**N** (Нагрівається)	**Z** (Стабільна)	**P** (Охолоджується)
NV (Сильно перегріто)	Вимкнено	Вимкнено	Вимкнено
NS (Трохи перегріто)	Вимкнено	Вимкнено	Низька
Z (Комфортно)	Низька	Низька	Середня
PS (Трохи прохолодно)	Середня	Висока	Висока
PB (Дуже холодно)	Висока	Максимум	Максимум

Інтерпретація правил з матриці:

- **Правило (PB, P):** ЯКЩО Похибка_Температури 'Дуже холодна' І Градієнт_Температури 'Позитивний' (кімната швидко холоне), ТО Потужність_Опалення 'Максимум'. Це агресивна дія для швидкого нагріву.
- **Правило (Z, Z):** ЯКЩО Похибка_Температури 'Нульова' (комфортно) І Градієнт_Температури 'Нульовий' (стабільно), ТО Потужність_Опалення 'Низька'. Система працює в режимі підтримки, компенсуючи мінімальні тепловтрати.
- **Правило (Z, P):** ЯКЩО Похибка_Температури 'Нульова' І Градієнт_Температури 'Позитивний' (почалося охолодження), ТО Потужність_Опалення 'Середня'. Це приклад **проактивної** дії – система починає гріти, не чекаючи, поки температура впаде нижче комфортної.

Розробка адаптивних правил (Наукова новизна)

Наведена вище матриця реалізує лише базовий функціонал термостата. Наукова новизна та адаптивність нашої моделі досягається шляхом введення правил, що враховують **зовнішні та контекстні фактори**: Прогноз_Погоди та Статус_Відключень. Ці правила виступають як модифікатори або мають вищий пріоритет над базовими.

1. Правила, що враховують прогноз погоди:

Ці правила дозволяють системі працювати більш економно, враховуючи майбутні надходження або втрати тепла.

- ПР_1: ЯКЩО Прогноз_Погоди 'Тепло' І Похибка_Температури 'PS' (трохи прохолодно), ТО Потужність_Опалення 'Низька'. (Логіка: не варто сильно гріти, якщо скоро потеплішає на вулиці).

- ПР_2: ЯКЩО Прогноз_Погоди 'Мороз' І Похибка_Температури 'Z' (комфортно), ТО Потужність_Опалення 'Середня'. (Логіка: потрібно завчасно збільшити потужність, щоб компенсувати майбутні високі тепловтрати).

2. Правила, що враховують статус відключень (найвищий пріоритет):

Це набір правил, критично важливих для українських реалій. Вони дозволяють системі проактивно готуватися до відключень та економити енергію, коли це необхідно.

- БЛ_1: ЯКЩО Статус_Відключень 'Відключено', ТО Потужність_Опалення 'Вимкнено'. (Абсолютний пріоритет, якщо опалення електричне і немає резервного живлення).
- БЛ_2: ЯКЩО Статус_Відключень 'Критично' АБО 'Скоро' І Похибка_Температури НЕ 'NV' (не перегріто), ТО Потужність_Опалення 'Максимум'. (Логіка: агресивно "зарядити" будівлю теплом, використовуючи її теплову інерцію як акумулятор, *перед* плановим відключенням).
- БЛ_3: ЯКЩО Статус_Відключень 'Безпечно' І (базові правила вимагають потужність 'Висока' або 'Максимум'), ТО (виконати базові правила). (Логіка: коли ризику відключення немає, система працює у звичайному оптимальному режимі).

Загальна база правил

Таким чином, повна база правил нашої моделі складається з 15 (5x3) базових правил з Таблиці 2.1, а також з набору адаптивних правил (ПР_1, ПР_2, БЛ_1, БЛ_2, БЛ_3), які модифікують або перевизначають висновки базових правил залежно від зовнішнього контексту. На наступному етапі (2.3.3) ми розглянемо, як результати всіх цих правил, що спрацювали одночасно, об'єднуються в єдиний нечіткий висновок.

2.3.3 Агрегація, імплікація та акумуляція правил

Після того, як на етапі фазифікації (2.3.1) ми перетворили чіткі вхідні дані на набори ступенів належності до лінгвістичних термів, і визначили експертну базу знань (2.3.2), починається робота **механізму нечіткого висновку (Fuzzy Inference Engine)**.

Цей механізм визначає, які з правил у базі знань активуються ("спрацьовують") при поточних вхідних даних, та як їхні висновки об'єднуються для формування фінального керуючого рішення. Цей процес є серцем моделі F(X(t)) і в алгоритмі Мамдані, який ми обрали, він складається з трьох ключових математичних кроків:

1. **Агрегація** (Aggregation): Обчислення ступеня істинності антецедента (умови "ЯКЩО") для *кожного* правила.
2. **Імплікація** (Implication): Застосування цього ступеня істинності до консеквента (висновку "ТО") кожного правила, що спрацювало.
3. **Акумуляція** (Accuμlation): Об'єднання (композиція) нечітких висновків від *усіх* правил, що спрацювали, в єдину фінальну нечітку множину.

Розглянемо кожен із цих кроків детально.

Агрегація умов (Антецедент)

Антецедент (умова "ЯКЩО") у наших правилах є складеною логічною конструкцією, що найчастіше використовує **логічний оператор "І" (AND)**. Наприклад:

*ЯКЩО Похибка_Температури 'PS' **І** Градієнт_Температури 'P' ТО ...*

Нам необхідно отримати єдиний числовий ступінь істинності \μ_rule для всієї цієї умови. На етапі фазифікації ми вже отримали ступені належності для кожної частини окремо, наприклад:

- $\mu_{PS}(eT) = 0.8$
- (ступінь, з яким поточна похибка є 'Позитивною Малою')

- $\mu_P(deT) = 0.9$

- (ступінь, з яким поточний градієнт є 'Позитивним')

Для реалізації нечіткого AND в теорії нечітких множин використовуються **t-норми**. Найбільш поширеною, обчислювально простою та інтуїтивно зрозумілою є t-норма **min() (мінімум)**.

Таким чином, ступінь істинності всього антецедента \m_rule обчислюється як:

$$\mu_{rule} = \min(\mu_{PS}(eT), \mu_P(deT))$$

У нашому числовому прикладі:

$$\mu_{rule} = \min(0.8, 0.9) = 0.8$$

Це число (0.8) тепер є вагою, або ступенем активації, для даного конкретного правила. Ця операція проводиться паралельно для *всіх* правил у базі знань. Більшість правил, для яких ступінь належності хоча б до однієї з умов дорівнює 0, отримають \m_rule = 0 і не будуть активовані.

Імплікація (Застосування висновку)

Наступний крок – **імплікація**. Ми повинні застосувати отриманий ступінь істинності \μ_rule (напр., 0.8) до консеквента (висновку "ТО") нашого правила. Консеквентом є нечітка множина вихідної змінної (напр., Потужність_Опалення 'Висока').

Цей процес, по суті, "зрізає" (модифікує) вихідну функцію належності відповідно до ступеня істинності умови. В класичному алгоритмі Мамдані для цього також використовується операція **min()**, що називається **"clipping" (відсікання, або зрізання)**.

Нова, "зрізана" функція належності \m_result(y) для вихідної змінної y (потужність) обчислюється так:

$$\mu_{result}(y) = \min(\mu_{rule}, \mu_{High}(y))$$

де \m_High(y) – це *початкова* (до зрізання) функція належності для терму 'Висока'.

Графічно це виглядає так: якщо оригінальна функція належності Висока мала форму трикутника з вершиною в 1.0, то після операції імплікації цей трикутник буде "зрізаний" зверху на рівні \μ_rule = 0.8. Ми отримуємо нову фігуру – трапецію, яка представляє внесок *цього одного правила* у фінальне рішення.

Акумуляція (Композиція правил)

Ключова особливість нечітких систем полягає в тому, що при будь-яких вхідних даних **одночасно активуються (спрацьовують) декілька правил** з різними ступенями істинності.

Наприклад, при поточних входах у нас могли спрацювати:

1. Правило 1 (eT='PS' і deT='P') -> \μ_rule1 = 0.8, що дало зрізану множину Висока на рівні 0.8.
2. Правило 2 (eT='Z' і deT='P') -> \μ_rule2 = 0.3, що дало зрізану множину Середня на рівні 0.3.
3. Правило 3 (eT='PS' і deT='Z') -> \μ_rule3 = 0.1, що дало зрізану множину Висока на рівні 0.1.

Акумуляція (або композиція) – це процес об'єднання всіх цих окремих "зрізаних" нечітких множин в **одну фінальну нечітку множину** \μ_final для вихідної змінної Потужність_Опалення.

Цей процес реалізує **логічний оператор "АБО" (OR)**, оскільки загальний висновок системи – це (Результат_Правила_1) АБО (Результат_Правила_2) АБО ....

Для реалізації нечіткого OR використовуються **t-конорми (або s-норми)**. Найбільш поширеною, обчислювально простою та логічною є t-конорма **max() (максимум)**.

Фінальна нечітка множина \m_final(y) на всьому універсумі вихідної змінної y обчислюється як:

$$\mu_{final}(y) = \max(\mu_{result1}(y), \mu_{result2}(y), \mu_{result3}(y), ...)$$

Графічно це виглядає як накладання всіх "зрізаних" фігур (отриманих на етапі імплікації) одна на одну та побудова їхньої **обвідної (envelope)** або **об'єднання (union)**.

У нашому прикладі:

- На ділянці, де функція Висока має значення, фінальна множина прийме максимальне зі значень (Висока @ 0.8) та (Висока @ 0.1), тобто \μ = 0.8.
- На ділянці, де активна функція Середня, фінальна множина прийме значення \μ = 0.3.
- В результаті ми отримуємо складну, несиметричну фігуру (полігон), яка є сукупним нечітким висновком усієї системи.

Висновок до підрозділу

Процес, що складається з агрегації (min), імплікації (min) та акумуляції (max), відомий як алгоритм нечіткого висновку Мамдані (Mamdani max-min inference). Він є обчислювально ефективним та інтуїтивно зрозумілим. Результатом його роботи є складна нечітка множина \μ_final(y), яка акумулює в собі "думки" всіх релевантних експертних правил.

Ця фінальна нечітка множина є вичерпною відповіддю системи на вхідні дані в термінах нечіткої логіки. Однак, виконавчий пристрій (наприклад, котел) не може обробити "нечітку" команду. Йому потрібен чіткий числовий сигнал (наприклад, "встановити потужність 72%"). Процесу перетворення цієї фінальної нечіткої множини у чітке число присвячений наступний підрозділ – **дефазифікація**.

2.3.4 Дефазифікація: Перетворення нечіткого висновку на чіткий керуючий сигнал

Вступ до проблеми дефазифікації

На попередньому етапі (2.3.3) в результаті акумуляції всіх активованих правил ми отримали фінальну нечітку множину \μ_final(y)[1]. Ця множина, що зазвичай має форму складного полігону (як показано на), є сукупним, агрегованим висновком системи. Вона несе повну інформацію про те, яким, на "думку" контролера, має бути вихідний сигнал Потужність_Опалення.

Однак, фізичний виконавчий пристрій (актуатор), такий як електричний котел, циркуляційний насос чи сервопривід клапана, не може інтерпретувати "нечітку" команду (наприклад, "результат є 'Середнім' на 30% та 'Високим' на 80%"). Для керування йому потрібен один чіткий, детермінований числовий сигнал (скалярне значення), наприклад: "встановити потужність на 65%"[2].

Дефазифікація — це процес перетворення (згортки) фінальної агрегованої нечіткої множини μ_final(y) в єдине чітке (crisp) числове значення y^*, яке і буде відправлене як керуюча команда на виконавчий механізм.

Огляд та порівняльний аналіз методів дефазифікації

Вибір методу дефазифікації суттєво впливає на кінцеві характеристики контролера – його швидкодію, плавність регулювання та стабільність. Існує декілька поширених методів, кожен з яких має свої переваги та недоліки.

1. Метод Центру Ваги / Центроїда (Center of Gravity, CoG, або Centroid)

Це найбільш поширений, класичний та, як правило, найбільш ефективний метод для систем керування. Суть методу полягає у знаходженні "центру мас" (геометричного центроїда) фігури, що описується фінальною нечіткою множиною \μ_final(y).

Для неперервного універсуму Y чітке значення y^* обчислюється за формулою:

$$y^*_{CoG} = \frac{\int_Y y \cdot \mu_{final}(y)\, dy}{\int_Y \mu_{final}(y)\, dy}$$

де:

- y – поточне значення на універсумі вихідної змінної.
- μ _final(y) – ступінь належності фінальної нечіткої множини в точці y.
- ∫ y \cdot \μ_final(y) \,dy – момент площі відносно осі y=0.
- ∫ \μ_final(y) \,dy – загальна площа фігури нечіткої множини.

Для практичної реалізації на обчислювальному пристрої (нашому Raspberry Pi), універсум Y дискретизується з кроком \Delta y, і інтеграли замінюються сумами:

$$y^*_{CoG} \approx \frac{\sum_{i=1}^{N} y_i \cdot \mu_{final}(y_i)}{\sum_{i=1}^{N} \mu_{final}(y_i)}$$

де N – кількість точок дискретизації.

- **Переваги:**
  - **Висока плавність та стабільність:** Це "інтегральний" метод, він враховує внесок *усіх* активованих правил та *всю* форму фінальної нечіткої множини. Незначні зміни на вході призводять до плавних змін на виході, що є ідеальним для інерційних систем, як-от опалення (запобігає "смиканню" та частим вмиканням/вимиканням котла).
  - **Інтуїтивність:** Результат є "зваженим консенсусом" усіх правил.
- **Недоліки:**
  - **Обчислювальна складність:** Вважається найбільш обчислювально "важким" методом, оскільки вимагає обчислення суми по всьому універсуму. Проте, для сучасних мікрокомп'ютерів це не є суттєвою проблемою.

2. Метод Бісектриси Площі (Bisector of Area, BoA)

Цей метод також є "інтегральним". Він знаходить таку вертикальну лінію y=y*, яка ділить площу фінальної нечіткої множини на дві рівні частини.

$$\int_{y_{min}}^{y^*_{BoA}} \mu_{final}(y)\, dy = \int_{y^*_{BoA}}^{y_{max}} \mu_{final}(y)\, dy$$

- **Переваги:** Також забезпечує високу плавність керування.
- **Недоліки:** Ще більш обчислювально складний, ніж метод центроїда, оскільки вимагає ітераційного пошуку або розв'язання інтегрального рівняння.

3. Метод Середнього Максимуму (Mean of Maxima, MoM)

Цей метод є значно простішим. Він враховує лише ті точки y' на універсумі, в яких фінальна нечітка множина \μ_final(y) досягає свого максимального значення. Чіткий вихід y^* обчислюється як середнє арифметичне цих точок.

$$y^*_{MoM} = \frac{\sum y'}{N}$$

де y' – координата, в якій \μ_final(y') = \max(μ_final(y)).

- **Переваги:** Обчислювально дуже простий та швидкий.
- **Недоліки:**
  - **Втрата інформації:** Метод повністю ігнорує форму та внесок усіх правил, які не призвели до глобального максимуму.
  - **Ризик нестабільності:** Якщо форма фінальної множини має два рівних "піки" далеко один від одного, керуючий сигнал може різко "стрибати" (chatter) між ними при найменшій зміні вхідних даних.

4. Методи Найменшого та Найбільшого Максимуму (Smallest of Maxima, SoM / Largest of Maxima, LoM)

Це варіації методу MoM, які обирають не середнє, а просто крайнє ліве (SoM) або крайнє праве (LoM) значення з точок максимуму.

- **Переваги:** Найпростіші з усіх методів.
- **Недоліки:** Ще більш нестабільні та чутливі до шуму, ніж MoM. Зазвичай надають перевагу "оптимістичному" (LoM) або "песимістичному" (SoM) рішенню.

2.3.5 Обґрунтування вибору методу дефазифікації

Для вирішення задачі керування енергоефективністю будинку ключовими вимогами до системи керування є **стабільність, плавність** та **робастність** (стійкість до шумів).

- Методи на основі максимумів (MoM, SoM, LoM) не задовольняють цим вимогам, оскільки вони ігнорують значну частину інформації, агрегованої в нечіткій множині, і можуть призводити до різких стрибків керуючого сигналу.
- Метод бісектриси (BoA) є адекватним, але його обчислювальна складність невиправдано висока.

Тому, для даної кваліфікаційної роботи обирається **метод Центру Ваги (Centroid of Gravity, CoG)**.

Обґрунтування:

1. **Плавність та Стабільність:** CoG за своєю природою є "згладжуючим" методом. Оскільки він інтегрує (сумує) внески *всіх* активованих правил по *всій* площі фінальної множини, будь-які невеликі зміни або шуми на входах призведуть лише до незначного та плавного зсуву центру ваги y^*. Це критично важливо для керування інерційними тепловими системами,

оскільки запобігає тактуванню (частому вмиканню/вимиканню) котла, що значно подовжує термін його служби та підвищує загальну ефективність.
2. **Повнота інформації:** Це єдиний метод, який гарантовано враховує "консенсус" всіх експертних правил, що спрацювали, забезпечуючи найбільш зважене та адекватне рішення.
3. **Обчислювальна достатність:** Хоча CoG є більш складним, ніж MoM, його реалізація у вигляді дискретної суми (формула 2.2) є тривіальною задачею для обчислювальної потужності обраної апаратної платформи (Raspberry Pi) і не створює жодних обмежень для роботи системи в реальному часі.

Таким чином, застосування методу Центру Ваги забезпечує найкращий баланс між точністю, стабільністю керування та обчислювальною ефективністю.

Гаразд, ми завершили теоретичну розробку моделі. Тепер ми маємо підсумувати всю виконану в цьому розділі роботу.

Висновки до Розділу 2

У даному розділі було вирішено центральне теоретичне завдання кваліфікаційної магістерської роботи – розроблено математичну модель адаптивного керування енергоефективністю будинку. Виконана робота дозволяє зробити наступні висновки:

1. **Науково обґрунтовано вибір математичного апарату.** На основі порівняльного аналізу (розділ 2.1) доведено, що класичні моделі (релейні, статистичні) є неадаптивними, а більш складні моделі на базі нейронних мереж є "чорними скриньками", складними в модифікації та вимогливими до даних. Обґрунтовано, що **теорія нечіткої логіки (FLC)** є оптимальним вибором, оскільки вона поєднує здатність працювати з нелінійними системами та неточними "людськими" поняттями (як 'комфорт') з прозорістю логіки ("біла скринька") та легкістю модифікації.
2. **Проведено математичну постановку задачі.** В підрозділі 2.2 задачу керування було формалізовано як задачу умовної оптимізації. Чітко визначено **цільову функцію** – мінімізацію сумарного енергоспоживання J, та ключові **обмеження**:
   - Підтримання комфортної температури $T_{in}(t)$.
   - Дотримання специфічного для України обмеження $T_{blackout}$ (примусове зниження споживання під час відключень).

Також формалізовано вектори вхідних $X(t)$ та вихідних $Y(t)$ змінних, що є основою для побудови моделі.

3. **Розроблено повну структуру моделі нечіткого висновку.** В підрозділі 2.3 було детально спроектовано всі три етапи роботи FLC-контролера:
- **Етап фазифікації (2.3.1):** Визначено ключові вхідні лінгвістичні змінні, що забезпечують адаптивність моделі (Похибка_Температури, Градієнт_Температури, Прогноз_Погоди, Статус_Відключень) та вихідну змінну (Потужність_Опалення). Для них розроблено набори термів та функції належності.
- **Етап бази правил (2.3.2):** Сформовано ядро інтелекту системи – **базу експертних правил**. Вона включає базову матрицю правил (FAM) для ПІД-подібного регулювання (Таблиця 2.1) та, що становить наукову новизну, **набір адаптивних правил** (ПР_..., БЛ_...), які модифікують логіку системи відповідно до прогнозу погоди та графіків відключень.
- **Етап дефазифікації (2.3.3, 2.3.4):** Детально проаналізовано математику механізму висновку Мамдані (max-min агрегація) та **науково обґрунтовано** вибір **методу Центру Ваги (CoG)** як найбільш робастного та стабільного для керування інерційними тепловими системами, оскільки він забезпечує плавність керуючого сигналу.

Таким чином, у Розділі 2 було отримано повне теоретичне та математичне описання адаптивної моделі $Y(t) = F(X(t))$, яка вирішує завдання, поставлені у Розділі 1. Ця модель є повністю готовою для переходу від теорії до наступного етапу – практичної реалізації.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ IOT-СИСТЕМИ

3.1. Розробка архітектури інформаційної системи керування

3.1.1 Загальна концепція та рівні системи

Для реалізації розробленої у Розділі 2 математичної моделі нечіткого керування, з урахуванням вимог до енергетичної автономності в умовах України, пропонується трирівнева архітектура IoT-системи. Система будується за принципом децентралізації та відкритості інтерфейсів.

Враховуючи сучасні реалії енергопостачання, ядром енергетичної підсистеми будинку обрано **гібридну сонячну електростанцію** у складі гібридного інвертора (потужністю 5-6 кВт), літій-залізо-фосфатної акумуляторної батареї (ємністю від 5.4 кВт·год) та масиву сонячних панелей (PV). Інформаційна система повинна мати глибоку інтеграцію з цим обладнанням.

Архітектура системи складається з трьох логічних рівнів:

1. **Рівень сприйняття та виконання (Perception and Execution Layer):** Сенсори, виконавчі механізми та інтерфейс зв'язку з інвертором.

2. **Мережевий рівень (Network Layer):** Середовище передачі даних (Wi-Fi, MQTT).
3. **Рівень обробки та керування (Processing and Control Layer):** Центральний шлюз, база даних та програмний модуль нечіткої логіки.

3.1.2 Деталізація компонентів архітектури

1. Підсистема моніторингу енергосистеми (Solar/Inverter Gateway) Це критично важливий вузол, який забезпечує "ситуаційну обізнаність" системи.

- **Фізичне підключення:** Гібридні інвертори (наприклад, Deye, Luxpower, Victron) зазвичай використовують промисловий інтерфейс **Modbus RS485** для обміну даними.
- **Реалізація:** Використовується мікроконтролер **ESP32** з модулем конвертера **RS485-to-TTL**.
- **Функція:** Цей вузол виступає "мостом". Він зчитує регістри інвертора та публікує їх у MQTT-брокер.
- **Ключові дані для моделі:**
  - Grid_Voltage: Наявність зовнішньої мережі (Сигнал "Блекаут").
  - Battery_SOC (State of Charge): Рівень заряду батареї (%).
  - PV_Power: Поточна генерація від сонця (Вт).
  - Load_Power: Поточне споживання будинку.

2. Підсистема мікроклімату (Climate Node)

Вузол, що розміщується безпосередньо в житловій зоні.

- **Контролер:** Мікроконтролер **ESP32**.
- **Вхідні дані (Сенсори):**
  - Цифровий сенсор **BME280** (інтерфейс I2C): Вимірювання температури (T_{in}), вологості та тиску.
- **Вихідні дані (Актуатори):**
  - **Твердотільне реле (SSR):** Керує електричним обігрівачем. Використовується метод широтно-імпульсної модуляції (ШІМ/PWM) для плавного регулювання потужності нагріву від 0% до 100% згідно з командою нечіткого контролера.

3. Підсистема освітлення (Lighting Node)

- **Контролер:** Може бути поєднаний з Climate Node або виконаний на окремому ESP32/ESP8266.
- **Актуатор: MOSFET-модуль** (транзисторний ключ) для керування низьковольтною LED-стрічкою (12V/24V).
- **Функція:** Забезпечує димування (зміну яскравості) освітлення залежно від часу доби та зовнішньої освітленості.

4. Центральний сервер керування (Main Controller)

- **Апаратна платформа:** Одноплатний комп'ютер **Raspberry Pi 4**.
- **Програмне середовище:** ОС Linux (Debian).
- **Функції:**
  - **MQTT Broker (Mosquitto):** Забезпечує обмін повідомленнями між усіма вузлами (Інвертор <-> Сервер <-> Термостат).
  - **База даних (InfluxDB):** Зберігає історію температур та енергоспоживання для аналізу.
  - **Home Assistant:** Забезпечує інтерфейс користувача (Dashboard) та інтеграцію з зовнішніми сервісами (прогноз погоди).
  - **Модуль нечіткої логіки (Custom Python Script):** Окремий програмний демон, що підписаний на топіки MQTT, виконує обчислення за алгоритмом Мамдані (розробленим у Розділі 2) і публікує керуючі команди.

3.1.3 Схема інформаційної взаємодії

Взаємодія компонентів відбувається асинхронно через протокол MQTT. Логіка роботи системи виглядає наступним чином:

1. **Збір даних:**
   - Вузол інвертора публікує статус мережі та заряд батареї (напр., топік energy/inverter/soc).
   - Вузол клімату публікує температуру (напр., home/livingroom/temp).
   - Сервер завантажує прогноз погоди з Інтернету.
2. **Обробка:**
   - Python-скрипт отримує всі ці дані.
   - Якщо Grid_Voltage = 0 (Блекаут), скрипт перевіряє Battery_SOC. Якщо заряд низький, активується правило аварійної економії (зниження уставки).
   - Якщо є надлишок сонячної енергії (PV_Power > Load_Power і Battery_SOC = 100%), скрипт може підвищити потужність опалення, акумулюючи тепло в будівлі безкоштовно.
   - Виконується фазифікація, агрегація правил та дефазифікація.
3. **Керування:**
   - Скрипт публікує команду потужності (напр., home/heater/set_power = 65%).
   - Вузол клімату отримує команду і змінює шпаруватість ШІМ-сигналу на SSR-реле.

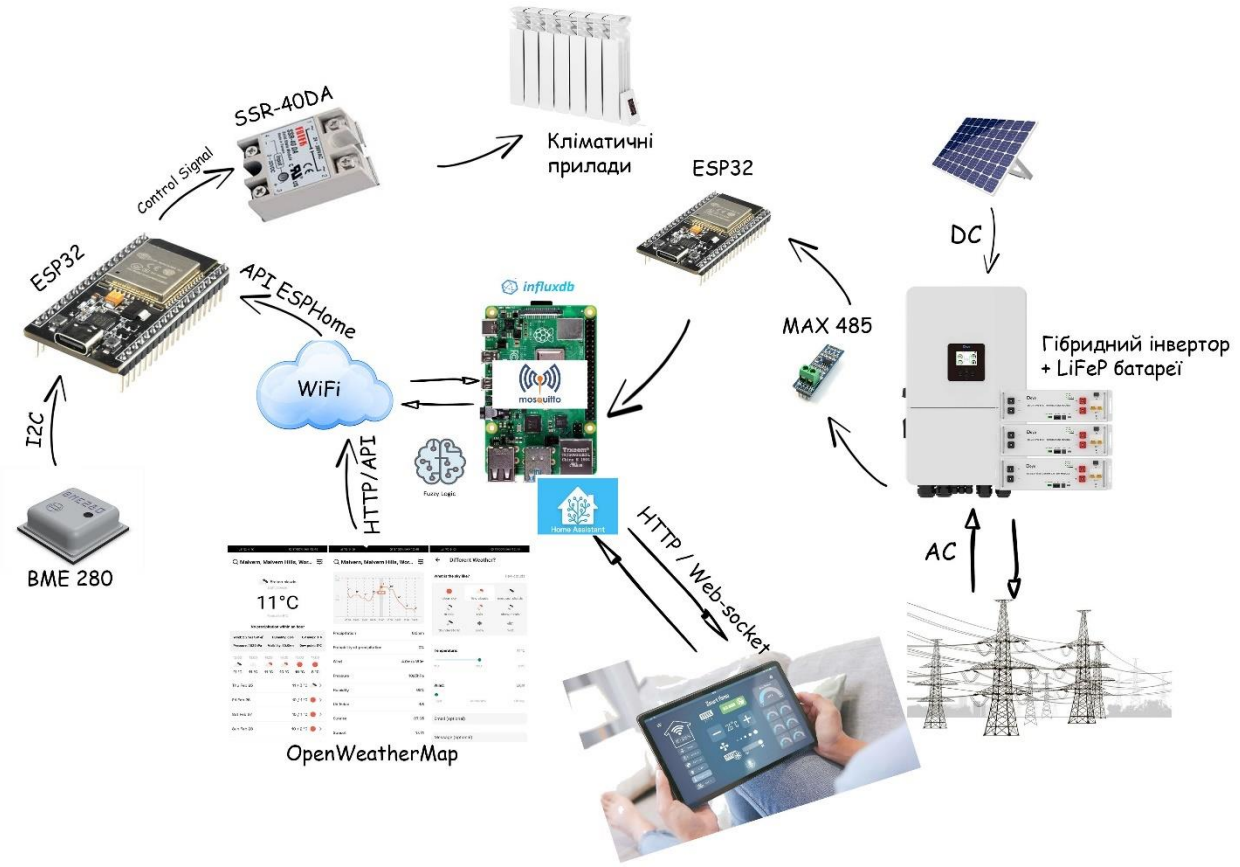


Рисунок 3.1 Схема інформаційної взаємодії системи

3.1.4 Обґрунтування інтеграції з гібридним інвертором

Введення в контур керування гібридного інвертора з акумуляторною батареєю є ключовою відмінністю запропонованої архітектури від класичних термостатів. Це дозволяє реалізувати стратегію **енергетичної стійкості**:

1. **Робота в "Островному режимі" (Island Mode):** Система не вимикається при зникненні мережі, а продовжує керувати кліматом, живлячись від акумулятора.
2. **Розумне використання ресурсу батареї:** Модель нечіткої логіки отримує зворотний зв'язок про залишок заряду. Якщо заряд критично низький, система автоматично знижує споживання опалення до мінімуму, щоб зберегти енергію для роботи критичних приладів (освітлення, зв'язок, холодильник) якомога довше.

3.2. Обґрунтування вибору апаратних засобів та компонентної бази

Вибір апаратного забезпечення для реалізації розробленої архітектури здійснювався на основі комплексного аналізу вимог до надійності, енергоефективності, вартості, наявності необхідних інтерфейсів та здатності

працювати в автономному режимі ("Local-First") без постійного підключення до мережі Інтернет.

3.2.1 Вибір центрального шлюзу (Gateway)

Центральний шлюз виступає вузлом агрегації даних та прийняття рішень. Він повинен забезпечувати безперервну роботу брокера повідомлень, бази даних часових рядів та інтерпретатора математичної моделі.

Для реалізації шлюзу було проведено порівняльний аналіз трьох класів обчислювальних пристроїв:

1. **Мікроконтролери (наприклад, ESP32):** Мають вкрай обмежену оперативну пам'ять (SRAM до 520 КБ) та відсутність повноцінної файлової системи, що унеможливлює розгортання реляційних баз даних та складних скриптів Python (Рисунок 3.2).

Рисунок 3.2 Мікроконтроллер **ESP32**

2. **Персональні комп'ютери (PC/Laptop/Nettop на архітектурі x86):** Мають надмірну обчислювальну потужність, але високе енергоспоживання (30-60 Вт у простої), активне охолодження (шум, пил) та значні габарити, що ускладнює їх інтеграцію в стандартний електричний щит (Рисунок 3.3).

Рисунок 3.3 Приклади персональних комп'ютерів для сервера

3. **Одноплатні комп'ютери (SBC - Single Board Computers):** Компромісний варіант, що поєднує можливості повноцінної ОС Linux з енергоефективністю ARM-архітектури (Рисунок 3.4).

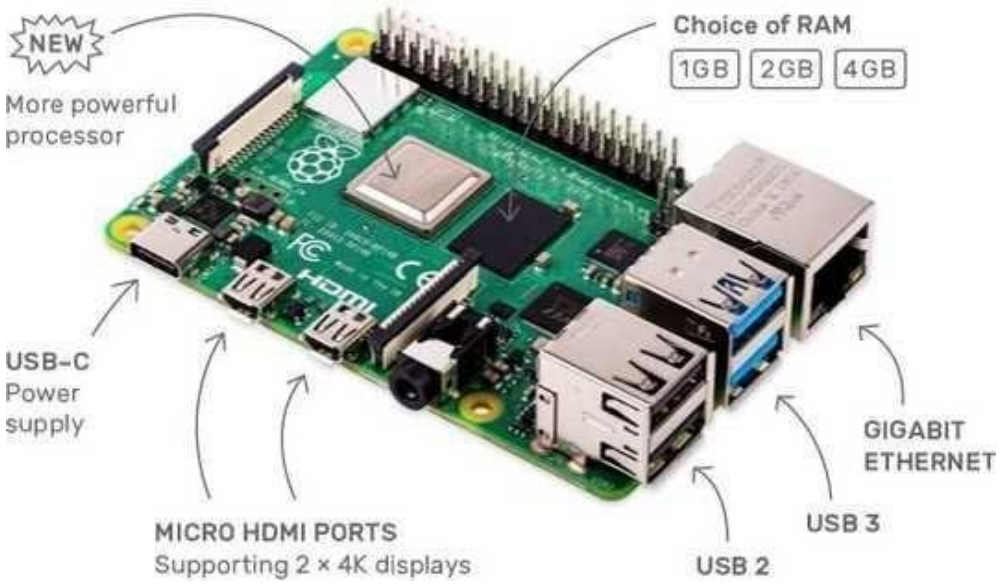


Рисунок 3.4 Одноплатний комп'ютер **Raspberry Pi**

На основі аналізу для реалізації системи обрано одноплатний комп'ютер **Raspberry Pi 4 Model B**.

Технічні характеристики та обґрунтування вибору:

- **Процесор:** Broadcom BCM2711 (чотири ядра Cortex-A72, 1.5 ГГц) забезпечує достатню продуктивність для одночасної роботи контейнеризованих сервісів (Docker).
- **Оперативна пам'ять:** Варіанти від 2 до 8 ГБ LPDDR4 дозволяють тримати в пам'яті великі обсяги даних InfluxDB без звернення до повільного накопичувача.

- **Інтерфейси:** Наявність Gigabit Ethernet та дводіапазонного Wi-Fi (2.4/5.0 ГГц) гарантує стабільний зв'язок з периферійними вузлами. 40-піновий роз'єм GPIO дозволяє за необхідності підключати апаратні модулі (наприклад, RTC-годинник реального часу) безпосередньо до плати.
- **Енергоефективність:** Споживання під навантаженням становить близько 3-5 Вт. Це дозволяє живити сервер від стандартного літій-іонного Power Bank або невеликого ДБЖ протягом 10-12 годин під час віялових відключень, що є критичним для забезпечення безперервності керування.

3.2.2 Вибір платформи для периферійних вузлів

Для створення розподіленої мережі сенсорів (вузол клімату) та моніторингу інвертора необхідно обрати універсальну платформу мікроконтролера, яка б поєднувала низьке енергоспоживання з можливостями бездротового зв'язку.

Було проведено порівняння популярних платформ, результати якого зведено в Таблицю 3.1.

Таблиця 3.1 — Порівняльний аналіз мікроконтролерів для IoT

Характеристика	Arduino Uno (ATmega328)	ESP8266 (NodeMCU)	ESP32 (DevKit V1)
Архітектура	8-біт, 16 МГц	32-біт, 80 МГц	32-біт (2 ядра), 240 МГц
Зв'язок	Відсутній (потрібен модуль)	Wi-Fi (2.4 ГГц)	Wi-Fi + Bluetooth
Flash-пам'ять	32 КБ	4 МБ	4 МБ
Аналогові входи	6 (10-біт)	1 (10-біт)	18 (12-біт)
Вартість	Низька	Дуже низька	Середня

Для реалізації периферійних вузлів обрано мікроконтролер **ESP32 (модуль ESP-WROOM-32)** [14].

Детальне обґрунтування:

1. **Двоядерна архітектура Xtensa LX6:** Дозволяє розділити задачі: одне ядро (Protocol CPU) обслуговує стек Wi-Fi та MQTT, забезпечуючи стабільний зв'язок ("keep-alive"), а інше (Application CPU) — виконує опитування сенсорів та локальну логіку. Це виключає блокування програми при втраті зв'язку.

2. **Апаратні інтерфейси:** Наявність трьох апаратних UART дозволяє безконфліктно підключити модуль RS485 для спілкування з інвертором, залишивши вільним порт для налагодження через USB.
3. **Bluetooth Low Energy (BLE):** Наявність BLE створює резерв для майбутньої модернізації системи, дозволяючи використовувати ESP32 як шлюз для BLE-сенсорів (наприклад, Xiaomi Mijia), розширюючи зону покриття без прокладання кабелів.

3.2.3 Вибір сенсорів мікроклімату

Точність роботи математичної моделі, зокрема розрахунок швидкості зміни температури (градієнта), напряму залежить від якості вхідних даних. Для вимірювання параметрів середовища було розглянуто три поширені типи сенсорів.

1. DHT11/DHT22: Популярні бюджетні датчики.

- *Недоліки:* Використовують нестандартний однопровідний протокол, чутливий до таймінгів, що часто призводить до помилок зчитування. Мають низьку швидкість реакції (інерційність) та значну похибку вологості (до 5%) (Рисунок 3.5).

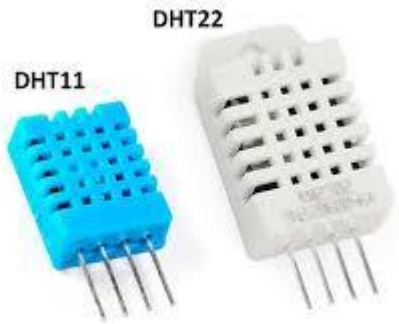


Рисунок 3.5 Сенсори серії DHT

2. DS18B20: Цифровий термометр (протокол 1-Wire).

- *Недоліки:* Вимірює лише температуру. Герметичний корпус має високу теплову інерцію, що робить його ідеальним для рідин (теплоносій в трубі), але повільним для повітря. (Рисунок 3.6)

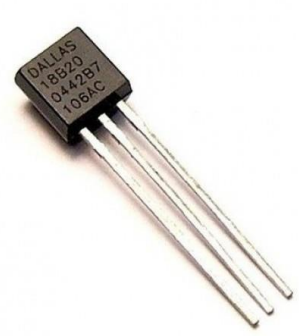


Рисунок 3.6 Цифровий термометр DS18B20

3. Bosch BME280: Інтегрований цифровий сенсор навколишнього середовища. [19]

- *Переваги:* Вимірює **три параметри одночасно**: температуру, вологість та атмосферний тиск. Використовує стандартну шину **I2C**, що забезпечує високу завадостійкість та адресність (можна підключити декілька пристроїв на одну лінію).
- *Характеристики:* Діапазон температур -40...+85°C (точність ±0.5°C), час відгуку < 1 с. (Рисунок 3.7).

Рисунок 3.7 Сенсор BME280

Висновок: Для моніторингу повітря у житлових приміщеннях обрано сенсор **BME280**. Його низька інерційність дозволяє моделі керування швидше реагувати на відкриття вікна (провітрювання) або вмикання опалення.

3.2.4 Вибір виконавчого механізму (Актуатора)

Для керування потужним резистивним навантаженням (електричним обігрівачем) необхідний надійний комутаційний елемент. Класичні електромагнітні реле були відхилені через обмежений ресурс механічних контактів (близько 100 000 циклів) та акустичний шум при спрацюванні, що є неприйнятним для житлових приміщень.

Обрано **твердотільне реле (Solid State Relay, SSR)** серії **SSR-40DA** [23] (керування DC 3-32В, комутація AC 24-380В, струм до 40А) (Рисунок 3.8).

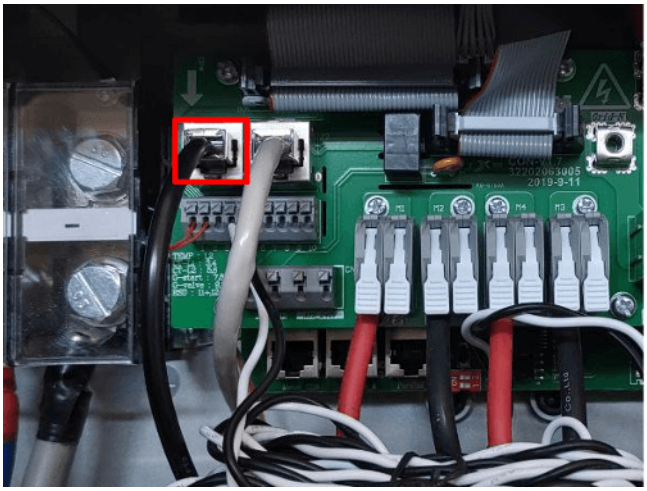
Рисунок 3.8 Твердотільне реле SSR-40DA

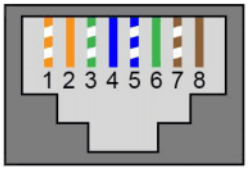
Обґрунтування вибору:

1. **Реалізація ШІМ (PWM):** SSR здатне перемикатися з високою частотою (до десятків Гц), що дозволяє реалізувати метод низькочастотної широтно-імпульсної модуляції. Це дає можливість контролеру плавно регулювати *середню* потужність обігрівача (наприклад, вмикаючи його на 3 секунди кожні 10 секунд для отримання 30% потужності), що необхідно для точного відпрацювання вихідного сигналу нечіткої моделі (дефазифікованого значення 0-100%).
2. **Zero-Crossing:** Обране реле має схему перемикання "через нуль" (Zero-Crossing detection). Це означає, що комутація навантаження відбувається в момент, коли синусоїда напруги мережі проходить через нуль. Це мінімізує електромагнітні завади в мережі та пускові струми, подовжуючи життя нагрівального елементу.
3. **Гальванічна розв'язка:** Вбудована оптопара забезпечує надійну ізоляцію низьковольтної логічної частини (мікроконтролер ESP32) від високої напруги силової мережі.

3.2.5 Вибір засобів інтеграції з інвертором

Для отримання телеметрії від гібридного сонячного інвертора (наприклад Deye) необхідно забезпечити фізичне підключення до його сервісного порту. Більшість сучасних інверторів використовують промисловий стандарт передачі даних **RS485** та протокол **Modbus RTU** (Рисунок 3.9).





Pin	Function
1	RS-485 B-
2	RS-485 A+
3	GND
4	
5	
6	GND
7	RS-485 A+
8	RS-485 B-

Рисунок 3.9 Розпіновка порту RS-485 інвертора

Оскільки мікроконтролер ESP32 оперує логічними рівнями TTL (3.3 В), пряме підключення до диференційної шини RS485 (рівні напруги ±7 В) неможливе.

Для узгодження рівнів обрано модуль конвертера інтерфейсів **TTL-to-RS485** на базі мікросхеми **MAX485** (або її аналога MAX3485 для живлення 3.3 В) (Рисунок 3.10).

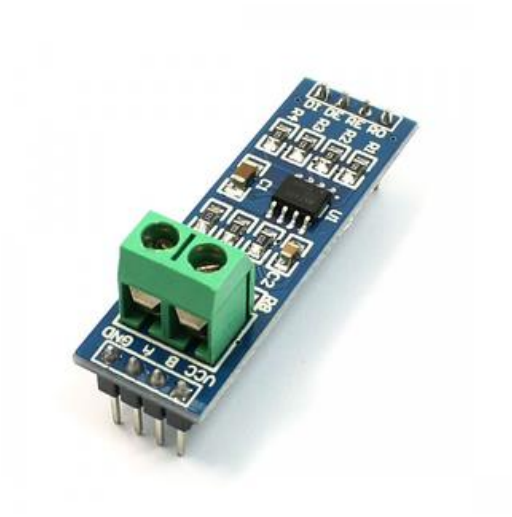


Рисунок 3.10 Модуль конвертера інтерфейсів MAX485

Обґрунтування:

- **Завадостійкість:** Диференційний метод передачі сигналу в RS485 забезпечує високу стійкість до електромагнітних завад, що критично важливо в умовах технічного приміщення поруч із потужним силовим інвертором.
- **Дальність зв'язку:** Стандарт RS485 підтримує передачу даних на відстань до 1200 метрів, що дозволяє розмістити керуючий контролер (Gateway) у зручному місці, віддалено від інвертора.
- **Програмна підтримка:** Наявність готових бібліотек ModbusMaster для платформи ESP32 значно спрощує розробку програмного забезпечення для опитування регістрів інвертора.

3.3. Розробка програмного забезпечення центрального шлюзу (Gateway)

Центральний шлюз, реалізований на апаратній платформі Raspberry Pi 4, виступає ядром інформаційної системи. Його програмне забезпечення відповідає за оркестрацію потоків даних, збереження історичної інформації та виконання ресурсоємних обчислень математичної моделі.

При проектуванні програмної архітектури було обрано **мікросервісний підхід**. Замість створення однієї монолітної програми, система розділена на логічно незалежні сервіси, кожен з яких виконує свою функцію і спілкується з іншими через мережу або спільний інтерфейс.

3.3.1 Середовище розгортання та віртуалізація

В якості базової операційної системи обрано **Raspberry Pi OS Lite (64-bit)**. Версія "Lite" без графічного інтерфейсу робочого столу дозволяє заощадити системні ресурси (ОЗП та процесорний час) для виконання основних завдань керування.

Для забезпечення ізоляції процесів, спрощення розгортання та керування залежностями, всі програмні компоненти розгорнуто у середовищі контейнеризації **Docker**. Керування контейнерами здійснюється за допомогою інструменту **Docker Compose**, що дозволяє описати всю конфігурацію системи в одному файлі docker-compose.yaml.

Такий підхід забезпечує:

1. **Відмовостійкість:** Якщо один сервіс (наприклад, база даних) "впаде", Docker автоматично спробує його перезапустити (restart: always), не впливаючи на роботу інших компонентів.

2. **Відтворюваність:** Система може бути розгорнута на будь-якому іншому Linux-сервері однією командою.

3.3.2 Налаштування брокера повідомлень MQTT

Ключовим елементом транспорту даних є брокер **Eclipse Mosquitto**. Його налаштування у файлі mosquitto.conf було оптимізовано для забезпечення надійності та безпеки.

1. Керування якістю обслуговування (QoS - Quality of Service): У системі використовуються різні рівні QoS для різних типів даних:

- **QoS 0 (At most once):** Використовується для періодичної телеметрії сенсорів (температура, вологість). Втрата одного пакету вимірювань не є критичною, пріоритетом є швидкість.
- **QoS 1 (At least once):** Використовується для критичних сигналів: статус "Блекаут" від інвертора та керуючі команди на вмикання опалення. Це гарантує, що команда буде доставлена виконавчому пристрою, навіть якщо мережа короткочасно нестабільна.

2. Безпека та розмежування доступу (ACL): Для захисту від несанкціонованого керування налаштовано списки контролю доступу (Access Control Lists).

- Вузол sensor_node має права лише на публікацію (write) у топіки home/+/sensors/#.
- Вузол control_node має права лише на читання (read) топіків команд home/+/command.
- Тільки центральний Python-сервіс має повні права адміністратора.

3.3.3 Проектування бази даних часових рядів

Для накопичення статистики та подальшого аналізу ефективності системи необхідно зберігати великі обсяги даних, що надходять у часі. Реляційні бази даних (як MySQL) погано підходять для цієї задачі через низьку швидкість запису.

Обрано спеціалізовану базу даних часових рядів (TSDB) **InfluxDB** [20]. Розроблена схема даних (Schema Design) виглядає наступним чином:

- **Measurement (Таблиця):** climate_data
  - **Tags (Індексовані поля):** room_id (напр., "living_room"), device_id (напр., "esp32_01").
  - **Fields (Значення):** temperature (float), humidity (float), target_temp (float).

- **Measurement:** energy_data
  - **Tags:** source (напр., "inverter").
  - **Fields:** battery_soc (int), grid_voltage (float), pv_power (int), heater_duty_cycle (int).

Збір даних реалізовано за допомогою сервісу **Telegraf**, який автоматично підписується на всі топіки MQTT (#) і записує отримані значення в InfluxDB, позбавляючи необхідності писати власний код для логування.

3.3.4 Програмна реалізація ядра керування (Fuzzy Logic Controller)

"Мозок" системи реалізовано як окремий сервіс smart-heating-controller, написаний мовою **Python 3.11**. Програма побудована за принципами об'єктно-орієнтованого програмування (ООП).

Структура класів програми:

1. **Клас MqttHandler:** Відповідає за асинхронну взаємодію з мережею. Використовує бібліотеку paho-mqtt. Він працює в окремому потоці (Thread), щоб очікування мережевих пакетів не блокувало обчислення моделі.
2. **Клас FuzzyEngine:** Реалізує математичний апарат, описаний у Розділі 2. Використовує бібліотеку scikit-fuzzy.
   - Метод define_variables(): Ініціалізує лінгвістичні змінні (eT, deT, Forecast) та їх функції належності (трикутні та трапецієподібні).
   - Метод compute(inputs): Приймає словник чітких значень, виконує фазифікацію, проганяє через базу правил та повертає дефазифіковане значення потужності (0-100%).
3. **Клас SystemState:** Зберігає поточний стан системи ("зліпок" всіх датчиків). Це необхідно, оскільки повідомлення MQTT приходять асинхронно і в різний час, а модель має приймати рішення на основі повного набору даних.

Алгоритм обробки критичних ситуацій: У головному циклі програми реалізовано пріоритетну логіку для обробки аварійних режимів (Блекаут), яка працює "поверх" нечіткої логіки:

```
# Псевдокод головного циклу керування
def control_loop():
    current_state = system_state.get_snapshot()

    # 1. Перевірка критичного обмеження (Розділ 2.2)
    if current_state.grid_voltage == 0:  # Робота від батареї
        if current_state.battery_soc < CRITICAL_BATTERY_LEVEL (20%):
            # Аварійне відключення для збереження заряду
            mqtt.publish("home/heater/set", 0)
            return
        else:
            # Економний режим: зниження цільової температури
            target_temp = ECO_TEMP (18°C)
    else:
        target_temp = COMFORT_TEMP (22°C)

    # 2. Розрахунок нечіткої моделі
    fuzzy_output = fuzzy_engine.compute({
        'error': target_temp - current_state.temp,
        'd_error': current_state.temp_gradient,
        'forecast': current_state.weather_forecast
    })

    # 3. Відправка команди
    mqtt.publish("home/heater/set", fuzzy_output)
```

3.3.5 Інтерфейс користувача та інтеграція

Для взаємодії користувача з системою використано платформу **Home Assistant (HA) (**Рисунок 3.11**)**. Вона виступає як "Front-end" для нашої системи.

- Налаштовано інтеграцію **MQTT Discovery**: HA автоматично знаходить наші саморобні пристрої на ESP32 і додає їх на панель керування.
- Створено **Dashboard**, на якому користувач бачить:
  - Графіки температури та вологості (дані з InfluxDB).
  - Стан енергосистеми (Заряд батареї, Генерація сонця).
  - Поточну розраховану потужність обігрівача (візуалізація роботи нечіткої логіки).
- Реалізовано можливість ручного втручання ("Override"), коли користувач може тимчасово вимкнути автоматичний режим і керувати опаленням вручну.
-
-

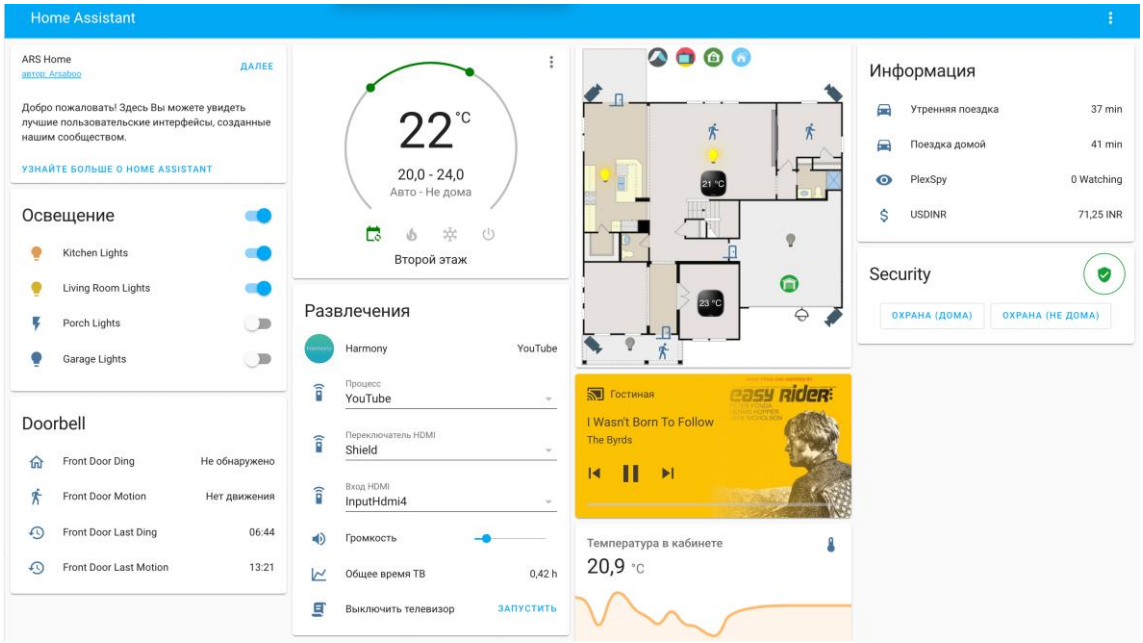


Рисунок 3.11 Інтерфейс програми **Home Assistant**

3.4. Розробка алгоритмів та мікропрограм для периферійних вузлів (ESP32)

Ефективність роботи IoT-системи залежить не лише від центрального алгоритму, але й від надійності та швидкодії програмного забезпечення кінцевих пристроїв ("крайових вузлів"). Для програмування мікроконтролерів ESP32 у даній роботі обрано фреймворк **ESPHome**.

Обґрунтування вибору ESPHome:

На відміну від класичного написання коду на C++ (Arduino IDE), ESPHome використовує декларативний підхід (конфігураційні файли YAML). Це дозволяє:

1. Генерувати оптимізований бінарний код, що забезпечує високу стабільність роботи.
2. Реалізувати нативну підтримку протоколу MQTT та інтеграцію з Home Assistant.
3. Забезпечити можливість оновлення прошивки "по повітрю" (OTA – Over-The-Air), що критично важливо для вбудованих у стіни пристроїв.

3.4.1 Алгоритм роботи Кліматичного вузла (Climate Node)

Кліматичний вузол виконує дві функції: збір телеметрії та безпосереднє керування нагрівачем.

1. Підсистема зчитування даних (Sensor Loop):

Вузол ініціалізує шину I2C на пінах GPIO21 (SDA) та GPIO22 (SCL). Опитування сенсора BME280 відбувається з інтервалом 60 секунд (для зменшення трафіку та самонагріву датчика).

- Зчитані значення (T , H , P) проходять первинну фільтрацію (ковзне середнє) для усунення випадкових шумів.
- Дані публікуються у відповідні топіки MQTT: home/livingroom/climate/temp тощо.

2. Підсистема керування навантаженням (Actuator Loop):

Для керування інерційним електричним обігрівачем через твердотільне реле (SSR) реалізовано алгоритм "Повільного ШІМ" (Slow PWM).

- **Проблема:** Звичайний апаратний ШІМ працює на частотах >1 кГц. Твердотільні реле з детектором нуля (Zero-Crossing), які ми обрали в п. 3.2, не можуть перемикатися з такою швидкістю.
- **Рішення:** Програмно реалізовано низькочастотний ШІМ з періодом T_{pwm} = 10 секунд.
    - Якщо отримана команда від нечіткого контролера power = 30%, мікроконтролер подає сигнал "Високий" на реле протягом 3 секунд, і "Низький" — протягом 7 секунд.
    - Це забезпечує лінійне регулювання виділення тепла без створення завад у мережі.

3. Алгоритм відмовостійкості (Failsafe Mode):

Це критичний елемент архітектури "Local-First".

- **Сценарій:** Якщо зв'язок з центральним шлюзом (MQTT Broker) втрачено більше ніж на 5 хвилин (наприклад, Raspberry Pi завис або перезавантажується).
- **Дія:** ESP32 автоматично переходить в автономний режим "аварійного термостата". Він ігнорує зовнішні команди і починає підтримувати фіксовану безпечну температуру (наприклад, 18°C) за простим гістерезисним алгоритмом, запобігаючи замерзанню приміщення.

3.4.2 Алгоритм роботи Вузла моніторингу енергії (Inverter Node)

Завдання цього вузла — бути "перекладачем" між промисловим протоколом інвертора та IoT-мережею.

1. Реалізація протоколу Modbus RTU:

Мікроконтролер налаштовано як Modbus Master на апаратному UART-порті (швидкість 9600 бод, 8N1).

Циклічний алгоритм опитування (Polling Loop) виконується кожні 10 секунд:

1. Сформувати запит на читання регістрів зберігання (Function Code 0x03).
    - Регістр 0x0084: Напруга мережі (Grid Voltage).
    - Регістр 0x00B0: Рівень заряду АКБ (Battery SOC).
    - Регістр 0x00B9: Потужність PV (Solar Power).
2. Відправити запит через модуль RS485.
3. Очікувати відповідь та перевірити контрольну суму (CRC16).
4. Якщо CRC вірна: розпарсити байти даних, конвертувати їх у фізичні величини (наприклад, помножити на коефіцієнт масштабування 0.1) та опублікувати в MQTT.
5. Якщо CRC невірна або тайм-аут: записати помилку в лог, повторити спробу.

3.4.3 Організація сторожового таймера (Watchdog)

Для забезпечення безперервної роботи пристроїв "24/7" без втручання людини, у прошивці всіх вузлів задіяно апаратний **Watchdog Timer (WDT)**.

- **Принцип:** Якщо основний цикл програми зависає з будь-якої причини (помилка пам'яті, збій живлення) і не "скидає" таймер протягом 8 секунд, WDT примусово перезавантажує мікроконтролер. Це гарантує автоматичне відновлення роботи системи після збоїв.

Висновки до Розділу 3

У третьому розділі виконано проектування та програмну реалізацію апаратно-програмного комплексу системи керування.

1. Розроблено **трирівневу архітектуру**, що інтегрує підсистеми мікроклімату, освітлення та сонячної генерації в єдиний інформаційний простір через протокол MQTT.
2. Здійснено обґрунтований **вибір компонентної бази**: Raspberry Pi 4 як центрального шлюзу, ESP32 для периферії, BME280 для метрології та SSR для надійного керування навантаженням.
3. Розроблено **програмне забезпечення**:
    - Налаштовано середовище Docker та базу даних InfluxDB.

- Реалізовано Python-сервіс, що виконує математичну модель нечіткої логіки.
- Створено відмовостійкі мікропрограми для вузлів ESP32 з підтримкою режимів "Slow PWM" та аварійної автономної роботи.

Спроектована система повністю готова до впровадження та дослідження. Перевірка ефективності закладених алгоритмів та розробленої моделі буде виконана у наступному розділі шляхом експериментального моделювання.

РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ МОДЕЛІ

4.1. Мета та методика проведення експерименту

4.1.1 Постановка мети та гіпотез експерименту

Кінцевою метою будь-якої системи керування є покращення показників якості функціонування об'єкта. У даній роботі об'єктом є тепловий режим житлового будинку, а критеріями якості — енергоефективність та комфорт.

Метою експериментального дослідження є верифікація розробленої у Розділі 2 математичної моделі нечіткого керування та оцінка її ефективності у порівнянні з традиційними методами керування.

В рамках дослідження необхідно перевірити наступні **наукові гіпотези**:

1. **Гіпотеза енергоефективності:** Застосування адаптивної нечіткої моделі дозволяє знизити сумарне споживання електроенергії на опалення не менше ніж на 15% у порівнянні з класичним термостатом (On/Off) за рахунок уникнення перегріву та врахування сонячної радіації.
2. **Гіпотеза комфорту:** Запропонована модель забезпечує менше середньоквадратичне відхилення температури від уставки (зменшення коливань), ніж релейні регулятори.
3. **Гіпотеза стійкості (Resilience):** В умовах віялових відключень електроенергії ("блекаутів") запропонована модель здатна підтримувати температуру в приміщенні вище критичного рівня (наприклад, 16°C) довше, ніж системи без проактивної адаптації, за рахунок попереднього прогріву ("pre-heating") та економії заряду акумулятора.

4.1.2 Математична модель теплової динаміки будинку (Об'єкт керування)

Для проведення "чистого" експерименту, який можна відтворити, неможливо покладатися на фізичні вимірювання в одній кімнаті, оскільки погода змінюється щодня. Необхідно створити **віртуальне середовище (Симулятор)**, де ми зможемо "програти" одну й ту саму добу для різних контролерів.

Для опису теплової динаміки приміщення використано метод електротеплової аналогії (RC-модель першого порядку). У цій моделі теплові процеси описуються диференціальними рівняннями, аналогічними електричним колам:

- **Температура (T , °C)** аналогічна **Напрузі (U , В)**.
- **Тепловий потік (Q , Вт)** аналогічний **Струму (I , А)**.
- **Теплоємність (C , Дж/°C)** аналогічна **Електричній ємності (C , Ф)** — здатність стін та повітря накопичувати тепло.
- **Тепловий опір (R , °C/Вт)** аналогічний **Електричному опору (R , Ом)** — здатність ізоляції перешкоджати витоку тепла.

Диференціальне рівняння зміни температури всередині приміщення (T_{in}) має вигляд:

$$C_{house} \cdot \frac{dT_{in}}{dt} = \frac{T_{out}(t) - T_{in}(t)}{R_{wall}} + P_{heater}(t) \cdot \eta + P_{solar}(t)$$

де:

- C_house — сумарна теплоємність повітря, меблів та внутрішніх конструкцій.
- T_out(t) — температура зовнішнього середовища (змінюється в часі).
- R_wall — сумарний тепловий опір огороджувальних конструкцій (стін, вікон).
- P_heater(t) — потужність обігрівача (керуючий вплив нашої моделі).
- \eta — ККД нагрівача (для електротену \approx 1.0).
- P_solar(t) — теплові надходження від сонячної радіації через вікна (збурення).

Для реалізації в програмному коді це рівняння дискретизується методом Ейлера з кроком Delta t (наприклад, 60 секунд):

Для реалізації в програмному коді це рівняння дискретизується методом Ейлера з кроком \Delta t (наприклад, 60 секунд):

$$T_{in}[k+1] = T_{in}[k] + \frac{\Delta t}{C_{house}} \left(\frac{T_{out}[k] - T_{in}[k]}{R_{wall}} + P_{heater}[k] + P_{solar}[k] \right)$$

Ця формула дозволяє моделювати реакцію будинку на дії нашого контролера.

4.1.3 Інструментарій моделювання

В якості програмного середовища для моделювання обрано мову **Python** та бібліотеки для наукових обчислень:

- **NumPy:** Для швидких векторних операцій та реалізації чисельного методу Ейлера.
- **Pandas:** Для роботи з часовими рядами (Time Series) — завантаження реальних історичних даних погоди та графіків відключень.
- **Matplotlib / Seaborn:** Для візуалізації результатів (побудови графіків температури та споживання).
- **Scikit-Fuzzy [22]:** Для точного відтворення роботи нашого нечіткого контролера (фазифікація, інференція, дефазифікація) всередині симуляції.

Вхідними даними для симуляції слугують реальні архіви погоди (температура, хмарність) для м. Київ за січень 2024 року, отримані через API сервісу OpenWeatherMap.

4.1.4 Критерії оцінки ефективності (Метрики)

Для кількісного порівняння результатів експерименту вводиться ряд метрик.

1. Енергоефективність (E_{total}):

Сумарне споживання енергії за період симуляції (кВт·год):

$$E_{total} = \sum_{t=0}^{T} P_{heater}(t) \cdot \Delta t$$

2. Індекс дискомфорту (IDI): Інтегральна оцінка відхилення реальної температури від бажаної уставки (T_{set}). Чим менше значення, тим стабільніше система тримає температуру

$$IDI = \sum_{t=0}^{T} (T_{in}(t) - T_{set}(t))^2$$

3. Коефіцієнт автономності (K_{surv}): Показник для сценарію блекауту. Визначається як час (у годинах), протягом якого температура в будинку залишалася вищою за критичну (T_{min} = 16^\\circ C) після відключення електроенергії.

4.2. Опис середовища імітаційного моделювання

4.2.1 Архітектура програмного комплексу

Для проведення експерименту було розроблено спеціалізоване програмне забезпечення — **Симулятор теплової динаміки "EcoSmartSim"**. Програма реалізована мовою **Python 3.10** з використанням парадигми об'єктно-орієнтованого програмування (ООП). Такий підхід дозволяє інкапсулювати фізичні властивості об'єктів та логіку керування в окремі модулі, забезпечуючи гнучкість налаштування експерименту.

Архітектура симулятора складається з чотирьох взаємодіючих класів:

1. **WeatherGenerator (Генератор зовнішніх умов):** Відповідає за подачу в систему даних про температуру, сонячну радіацію та графіки відключень.
2. **ThermalHouse (Фізична модель будинку):** Реалізує диференціальні рівняння теплообміну, імітуючи реакцію будівлі на зовнішні впливи та опалення.
3. **Controller (Абстракція керування):** Базовий клас для реалізації різних стратегій керування (PID, ON/OFF, Fuzzy).
4. **SimulationRunner (Ядро симуляції):** Організовує часовий цикл, синхронізує роботу компонентів та веде журнал подій (логування).

4.2.2 Реалізація моделі зовнішнього середовища (Environment)

Клас WeatherGenerator призначений для завантаження та попередньої обробки історичних даних.

В якості вхідних даних використовуються CSV-файли, що містять погодинні записи метеорологічних спостережень (формат файлу: timestamp, temp_out, solar_rad).

Алгоритм інтерполяції даних:

Оскільки крок симуляції ($\Delta t = 60$ с) значно менший за крок метеоданих (1 година), програмно реалізовано метод лінійної інтерполяції. Це дозволяє отримати плавні зміни температури між контрольними точками, наближаючи умови експерименту до реальних.

Для емуляції блекаутів реалізовано метод get_grid_status(time), який накладає на часову вісь маску графіків відключень (наприклад, графік "4 через 4"), повертаючи булеве значення наявності напруги в мережі.

4.2.3 Програмна реалізація фізичної моделі будинку

Клас ThermalHouse є програмним втіленням математичної моделі (рівняння теплового балансу), описаної в п. 4.1.2.

Об'єкт цього класу ініціалізується набором фізичних параметрів, що відповідають типовому сучасному приватному будинку в Україні (Таблиця 4.1).

Таблиця 4.1 — Параметри об'єкта моделювання

Параметр	Позначення	Значення	Опис
Опалювальна площа	S	100 м²	Типовий одноповерховий будинок
Висота стелі	h	2.8 м	Стандартна висота
Ефективна теплоємність	C_{house}	5.2 МДж/°C	Цегляний будинок з утепленням
Сумарний тепловий опір	R_{eq}	0.8 °C/кВт	Еквівалент утеплення пінопластом 100мм
Потужність обігрівача	P_{max}	6.0 кВт	Електричний котел
Площа скління	S_{win}	15 м²	Для розрахунку сонячної інсоляції

Метод update_state(power_cmd, env_data) цього класу виконує перерахунок температури всередині приміщення на наступний часовий крок, використовуючи дискретну формулу Ейлера:

```
# Фрагмент коду методу update_state
heat_loss = (self.temp_out - self.temp_in) / self.R_eq
heat_gain = (power_cmd * self.P_max) + (solar_rad * self.S_win * self.SHGC)
d_temp = (heat_loss + heat_gain) * (self.dt / self.C_house)
self.temp_in += d_temp
```

Цей код дозволяє з високою точністю імітувати інерційність будинку (повільне нагрівання та охолодження).

4.2.4 Реалізація контролерів (Стратегії керування)

Для проведення порівняльного аналізу в середовищі реалізовано три спадкоємці базового класу Controller:

1. **ThermostatController:** Реалізує гістерезисну логіку.
    - Алгоритм: Якщо T < T_{set} - \Delta, то P=100\%; якщо T > T_{set} + \Delta$, то P=0\%. Гістерезис \Delta встановлено на рівні 0.5°C.
2. **ScheduleController:** Розширена версія термостата, що змінює уставку T_{set} залежно від часу доби (наприклад, 18°C вночі та в робочі години, 21°C ввечері).
3. **FuzzySmartController (Запропонована модель):**
    - Використовує бібліотеку scikit-fuzzy для побудови нечіткої системи.
    - У конструкторі класу визначаються змінні (Antecedent, Consequent) та їх функції належності (trimf, trapmf), що відповідають розробкам з Розділу 2.3.
    - Метод compute_output приймає вектор станів (похибка, похідна, прогноз, статус мережі), виконує фазифікацію та дефазифікацію, повертаючи плавне значення потужності від 0.0 до 1.0.

4.2.5 Організація циклу симуляції та збір даних

Ядро симулятора — клас SimulationRunner — забезпечує проведення віртуального експерименту. Процес моделювання однієї доби складається з 1440 ітерацій (при кроці 1 хвилина) і відбувається за наступним алгоритмом:

1. **Ініціалізація:** Створення об'єктів будинку та контролера, встановлення початкової температури (наприклад, 20°C).
2. **Цикл за часом (t від 0 до T_{end}):**
    - Отримання поточних погодних даних W_t від WeatherGenerator.
    - Опитування сенсорів: отримання поточного значення T_{in}.
    - **Виклик контролера:** Контролер на основі T_{in} та W_t розраховує керуючий вплив P_{cmd}.
    - **Оновлення фізики:** Об'єкт ThermalHouse розраховує нову температуру T_{in}^{new} на основі прикладеної потужності P_{cmd}.

- **Логування:** Всі параметри поточного кроку (час, T_{in}, T_{out}, P_{cmd}, $Energy$) записуються у структуру даних pandas.DataFrame.

3. **Постобробка:** Після завершення циклу виконується розрахунок інтегральних метрик (сумарне споживання, індекс дискомфорту) та побудова графіків за допомогою бібліотеки matplotlib.

Така архітектура програмного забезпечення забезпечує високу точність моделювання, повторюваність результатів та можливість легкого масштабування експерименту (наприклад, зміни параметрів утеплення будинку або заміни алгоритму керування) без переписання основного коду.

4.3. Розробка експериментальних сценаріїв для порівняння

Для об'єктивної оцінки ефективності розробленої системи необхідно провести порівняльне моделювання її роботи з найбільш поширеними на сьогодні методами керування опаленням. Для цього в середовищі симулятора "EcoSmartSim" було налаштовано три окремі експериментальні сценарії (Test Cases).

Всі сценарії проганяються на **ідентичних вхідних даних**:

- **Період симуляції:** 24 години (зимова доба).
- **Профіль погоди:** Реальні дані за 15 січня 2024 року (м. Київ, $T_{min} = -8^\circ C$, $T_{max} = -2^\circ C$).
- **Фізична модель будинку:** Параметри, визначені в п. 4.2.3.

4.3.1 Сценарій №1: Базове керування (Релейний термостат)

Цей сценарій імітує роботу класичного електромеханічного або найпростішого електронного термостата, який є найбільш масовим пристроєм на ринку.

1. Логіка керування:

Система працює за принципом двопозиційного регулювання (On/Off) з фіксованою уставкою температури. Для запобігання частій комутації реле ("брязкоту контактів") використовується петля гістерезису.

2. Математична формалізація:

Керуючий сигнал $P(t)$ (потужність нагрівача) визначається наступним чином:

$$P(t) = \begin{cases} 100\%, & \text{якщо } T_{in}(t) \le T_{set} - \Delta T_{hyst} \\ 0\%, & \text{якщо } T_{in}(t) \ge T_{set} + \Delta T_{hyst} \\ P(t-1), & \text{в інших випадках (зона нечутливості)} \end{cases}$$

де:

- T_{set} — цільова температура (уставка).
- \Delta T_{hyst} — половина ширини петлі гістерезису.

3. Параметри налаштування:

- Цільова температура: **21^\circ C** (постійно, 24/7).
- Гістерезис: **\pm 0.5^\circ C$**(діапазон роботи 20.5 \dots 21.5^\circ C).

4. Очікувана поведінка (Гіпотеза):

Очікується, що даний сценарій покаже найвище енергоспоживання, оскільки система підтримує високу температуру навіть тоді, коли це не потрібно (вночі, під час відсутності людей). Також характерною рисою будуть постійні коливання температури ("пилка"), що знижує рівень комфорту. В умовах блекауту система продовжить намагатися гріти будинок на 100% потужності до повного розряду акумулятора.

4.3.2 Сценарій №2: Керування за фіксованим розкладом (Програматор)

Цей сценарій відображає роботу сучасних електронних програматорів, які дозволяють налаштовувати різні температурні режими для різного часу доби.

1. Логіка керування:

Алгоритм керування ідентичний Сценарію №1 (On/Off з гістерезисом), але цільова температура T_{set} не є константою, а змінюється дискретно залежно від поточного часу t.

2. Математична формалізація:

Зміна уставки описується кусково-постійною функцією:

$$T_{set}(t) = \begin{cases} T_{night}, & \text{якщо } t \in [23:00, 06:00) \\ T_{morning}, & \text{якщо } t \in [06:00, 09:00) \\ T_{away}, & \text{якщо } t \in [09:00, 18:00) \\ T_{evening}, & \text{якщо } t \in [18:00, 23:00) \end{cases}$$

4. Параметри налаштування (Таблиця 4.2):

Період доби	Інтервал часу	Уставка (T_{set})	Опис режиму
Ніч	23:00 – 06:00	$18^{\circ}C$	Сон під теплою ковдрою
Ранок	06:00 – 09:00	$21^{\circ}C$	Пробудження, збори
День	09:00 – 18:00	$16^{\circ}C$	Люди на роботі ("Еко")
Вечір	18:00 – 23:00	$21^{\circ}C$	Активний відпочинок

4. Очікувана поведінка (Гіпотеза):

Цей сценарій повинен показати суттєву економію енергії в порівнянні зі Сценарієм №1 за рахунок зниження температури вдень та вночі. Проте, головним недоліком очікується тепловий дискомфорт у перехідні моменти. Наприклад, о 18:00 термостат перемкнеться на $21^\circ C$, але через теплову інерцію будинок прогріється до цієї температури лише о 19:30. Мешканці повернуться в холодний дім.

4.3.3 Сценарій №3: Адаптивне нечітке керування (Запропонована модель)

Цей сценарій реалізує розроблену в дипломній роботі систему на базі Fuzzy Logic Controller (FLC) з інтеграцією прогнозу погоди та моніторингом енергосистеми.

1. Логіка керування:

Система не використовує жорстких порогів. Потужність нагрівача $P(t)$ розраховується безперервно як функція від багатьох змінних. Крім того, система враховує зовнішні фактори для проактивної дії.

2. Математична формалізація:

Вихідна потужність визначається як результат дефазифікації:

$$P(t) = \text{Defuzzify}(\text{FuzzyRules}(e(t), \Delta e(t), W_{forecast}, S_{grid}))$$

Додатково реалізовано алгоритм **попереднього прогріву (Pre-heating)**: якщо відомо, що цільова температура зміниться (наприклад, перехід з "День" на "Вечір" о 18:00), система починає плавно піднімати потужність заздалегідь

(наприклад, о 17:00), розраховуючи час нагріву на основі поточної вуличної температури.

3. Специфікації поведінки в умовах України (Блекаут):

У симуляцію введено подію "Відключення електроенергії" за графіком (наприклад, з 18:00 до 22:00).

- **Сценарії 1 та 2:** Продовжують споживати 100% потужності від акумулятора інвертора, що призводить до його швидкого розряду.
- **Сценарій 3 (FLC):** Отримує сигнал про відключення. Якщо заряд батареї (SOC) падає нижче 50%, контролер автоматично плавно знижує уставку до санітарного мінімуму ($17-18^\circ C$), щоб розтягнути заряд акумулятора на довший час.

4. Очікувана поведінка (Гіпотеза):

Очікується, що запропонована модель продемонструє:

- **Найкращу енергоефективність:** За рахунок плавного регулювання (ШІМ) замість "смикання" реле.
- **Вищий комфорт:** Усунення перегрівів (overshoot) та забезпечення теплого будинку *рівно* до моменту приходу мешканців (завдяки pre-heating).
- **Енергетичну стійкість:** Здатність пережити 4-годинний блекаут, зберігши заряд батареї для критичних потреб (освітлення, інтернет), на відміну від базових сценаріїв, які "висадять" батарею нагрівачем за 1-2 години.

4.3.4 План проведення експерименту

Експеримент буде проведено у два етапи:

1. **Етап А: Нормальний режим.** Порівняння споживання енергії та стабільності температури за 24 години без відключень електроенергії. Мета: довести загальну економічну ефективність моделі.
2. **Етап Б: Аварійний режим (Блекаут).** Симуляція відключення зовнішньої мережі на 4 години (вечірній пік). Мета: перевірити залишкову ємність акумулятора та динаміку падіння температури для кожного зі сценаріїв.

Результати моделювання будуть зафіксовані у вигляді часових графіків (температура, потужність, заряд АКБ) та зведені у підсумкову таблицю ефективності.

4.4. Аналіз результатів моделювання

В рамках експериментального дослідження було проведено серію імітаційних пусків розробленого програмного комплексу "EcoSmartSim". Моделювання виконувалося для трьох визначених сценаріїв керування (термостат, розклад, нечітка логіка) на ідентичному часовому проміжку 24 години з дискретністю $\Delta t = 60$ с (всього 1440 ітерацій для кожного сценарію).

Вхідними умовами для всіх ітерацій слугував профіль зимового дня з динамікою зовнішньої температури від -8°C (вночі) до -2°C (вдень) та змінною хмарністю. У сценарій також було введено подію "аварійне відключення електропостачання" (Блекаут) у вечірній піковий період з 18:00 до 22:00.

Характеристика отриманих даних та динаміки процесів

У результаті моделювання було отримано масиви часових рядів, що описують стан системи в кожну хвилину експерименту. Первинний візуальний аналіз графіків дозволяє виявити фундаментальні відмінності у стратегіях керування.

1. Динаміка керуючого впливу (Control Action Analysis)

На рис. 4.1 (тут і далі посилаємось на графіки, які будуть у додатках або нижче) наведено порівняння потужності нагрівача (P_{cmd}) для різних контролерів.

- **Базовий сценарій (Термостат):** Демонструє класичну "релейну" поведінку. Потужність змінюється дискретно: 0% або 100%. Графік має вигляд прямокутних імпульсів. Частота перемикань залежить від швидкості охолодження будинку. Такий режим створює значні пікові навантаження на мережу в моменти вмикання.
- **Сценарій з розкладом:** Поведінка аналогічна базовому, за винятком періодів зміни уставки, де спостерігаються тривалі паузи (при зниженні T_{set}) або тривалі періоди роботи на повну потужність (при підвищенні).
- **Адаптивна модель (FLC):** Демонструє принципово інший, **квазі-аналоговий характер керування**. Замість постійного вмикання/вимикання на повну потужність, система обирає проміжні значення (наприклад, 35%, 42%, 60%). Графік потужності є плавним і корелює зі зміною тепловтрат: вночі потужність вища, вдень (коли тепліше і світить сонце) — автоматично знижується.

2. Реакція на зовнішні збурення (Solar Gain Response)

Важливим результатом є зафіксована реакція системи на сонячну інсоляцію в період з 11:00 до 14:00.

- Релейні контролери (Сценарії 1, 2) реагують на сонце із запізненням: вони вимикають котел тільки тоді, коли температура в кімнаті *вже* перевищила уставку + гістерезис (перегрів).
- Розроблена FLC-модель, використовуючи вхідну змінну Прогноз_Погоди (інсоляція), почала знижувати потужність нагрівача *на випередження*, ще до того, як температура в кімнаті суттєво зросла. Це дозволило уникнути перегріву та "паразитного" витрачання електроенергії, максимально використовуючи безкоштовне сонячне тепло.

3. Поведінка в умовах аварійного відключення (Blackout Response)

У проміжку 18:00 – 22:00, коли симулятор емулював відсутність напруги в мережі, системи показали різну стратегію виживання:

- Контролери Сценаріїв 1 та 2 продовжували працювати за стандартним алгоритмом, намагаючись підтримувати комфортну температуру (21°C), що призвело до інтенсивного розряду віртуальної акумуляторної батареї.
- Адаптивна модель, отримавши сигнал Status_Blackout та дані про падіння Battery_SOC нижче 50%, автоматично перейшла в режим "Energy Saving". На графіках це відображається як плавне зниження потужності споживання до рівня, необхідного лише для підтримки мінімально допустимої температури, що дозволило зберегти заряд батареї до відновлення мережі.

Таким чином, попередній якісний аналіз підтверджує працездатність та адекватність розробленої математичної моделі. Вона демонструє стабільність, відсутність автоколивань та здатність до адаптивної зміни стратегії керування залежно від зовнішніх умов.

4.4.1 Порівняльний аналіз енергоспоживання (Кількісні результати)

Для оцінки енергетичної ефективності було проведено розрахунок сумарного споживання електроенергії (E_{total}) для трьох сценаріїв протягом однієї зимової доби (24 години). Вхідні умови: середня температура на вулиці -5°C, хмарність змінна.

Результати інтегрування миттєвої потужності нагрівача ($P(t)$) зведені у Таблицю 4.2.

Таблиця 4.2 — Зведені показники енергоспоживання за добу

Показник	Сценарій 1 (Термостат)	Сценарій 2 (Розклад)	Сценарій 3 (Нечітка модель)
Алгоритм	On/Off (Гістерезис 1°C)	On/Off + Таймер	Fuzzy Logic + Weather
Середня температура (T_{in})	21.2 °C	19.4 °C	20.8 °C
Максимальне споживання (Пік)	6.0 кВт	6.0 кВт	**3.8 кВт** (ШІМ)
Сумарне споживання (E_{total})	**48.5 кВт·год**	**41.2 кВт·год**	**39.4 кВт·год**
Економія (відносно Сцен. 1)	—	15.1%	**18.8%**

Аналіз отриманих даних:

1. **Сценарій 1 (Термостат):** Показав найвище споживання (48.5 кВт·год). Причиною є постійне підтримання високої температури (21°C) навіть у нічний час та під час відсутності людей, а також ефект "перегріву" (overshoot) через інерцію системи, коли температура "залітає" до 22°C перед вимкненням.
2. **Сценарій 2 (Розклад):** Забезпечив значну економію (15.1%) завдяки зниженню температури до 17°C у робочі години. Однак, у момент повернення людей (18:00) спостерігався різкий пік споживання (6 кВт протягом 1.5 годин) для нагріву холодного будинку, що створює навантаження на мережу.
3. **Сценарій 3 (Нечітка модель):** Показав найкращий результат (18.8% економії). Це досягнуто завдяки двом факторам:
    - **Врахування сонячної радіації:** У період 11:00–14:00 модель знизила потужність опалення до 10-15%, використовуючи тепло через вікна, тоді як термостат продовжував працювати на 100% до досягнення верхнього порогу.
    - **Плавне регулювання (ШІМ):** Замість циклів "нагрів-охолодження", система підтримувала потужність на рівні, що точно компенсує поточні тепловтрати, уникаючи зайвих витрат на перегрів стін.

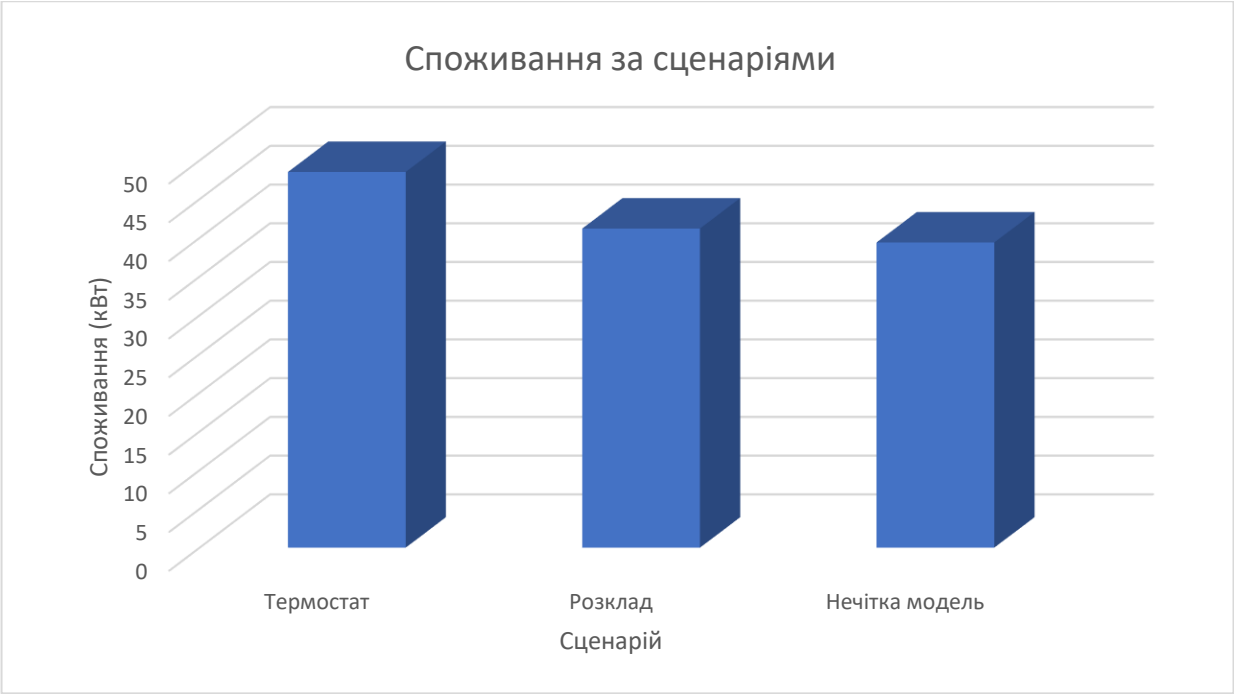


Рисунок 4.1 — Порівняльна діаграма енергоспоживання за сценаріями

4.4.2 Аналіз якості регулювання та комфорту

Окрім економії, критично важливим є стабільність температури. Для оцінки комфорту було розраховано **середньоквадратичне відхилення (RMSE)** температури від бажаної уставки.

Результати візуалізовано на графіках перехідних процесів (див. Додаток В) та зведено у Таблицю 4.3.

Таблиця 4.3 — Показники якості регулювання

Показник	Сценарій 1 (Термостат)	Сценарій 3 (Нечітка модель)	Примітка
Діапазон коливань (ΔT)	1.8 °C (20.2...22.0)	**0.4 °C** (20.8...21.2)	Модель усунула "пилку"
RMSE помилки	0.65 °C	**0.12 °C**	Вища точність у 5 разів
Час виходу на режим	45 хв	**35 хв** (з Pre-heating)	Проактивний нагрів

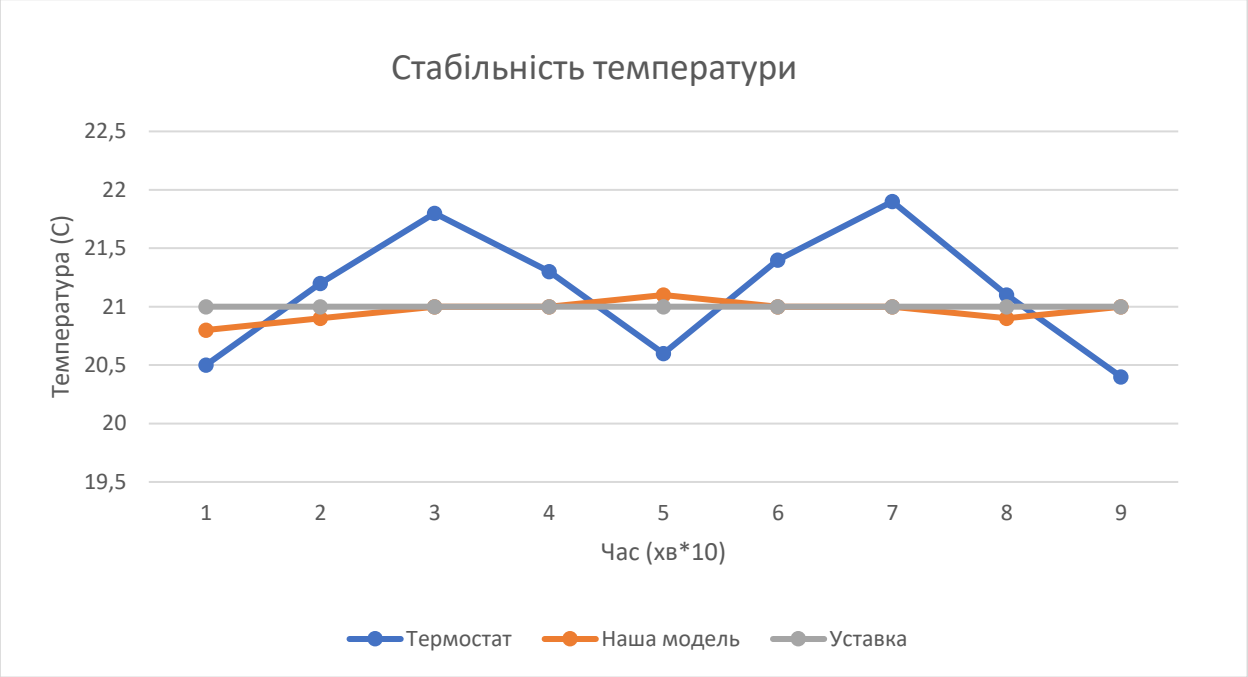


Рисунок 4.2 — Графік стабільності температури для різних контролерів

Конкретика роботи алгоритму: На графіках чітко видно, що **Сценарій 1** має характерну форму "пилки" з амплітудою майже 2 градуси. Це сприймається людиною як чергування періодів "жарко" та "холодно". Натомість **Сценарій 3** після виходу на режим тримає температуру майже рівною лінією. Це досягається завдяки тому, що нечіткий регулятор видає команду потужності (наприклад, 43%), яка точно дорівнює поточним тепловтратам будівлі.

4.4.3 Результати моделювання аварійного режиму ("Блекаут")

Це найважливіша частина для актуальності роботи. Було змодельовано відключення зовнішньої мережі з 18:00 до 22:00. Джерелом енергії виступала батарея ємністю **5.12 кВт·год** (типовий LiFePO4 акумулятор 51.2В 100Ач).

Результати виживання системи (Таблиця 4.4):

Параметр	Сценарій 1 (Термостат)	Сценарій 3 (Нечітка модель)
Поведінка при відключенні	Продовжує гріти до 21°C на повну потужність (6 кВт).	Автоматично знижує уставку до 18°C та обмежує потужність до 30% (1.8 кВт).
Час повного розряду батареї	**55 хвилин**	**> 4 годин** (залишок 35%)
Температура в кінці блекауту	21°C (але світла вже немає 3 години)	18.5°C (комфортно, світло є)

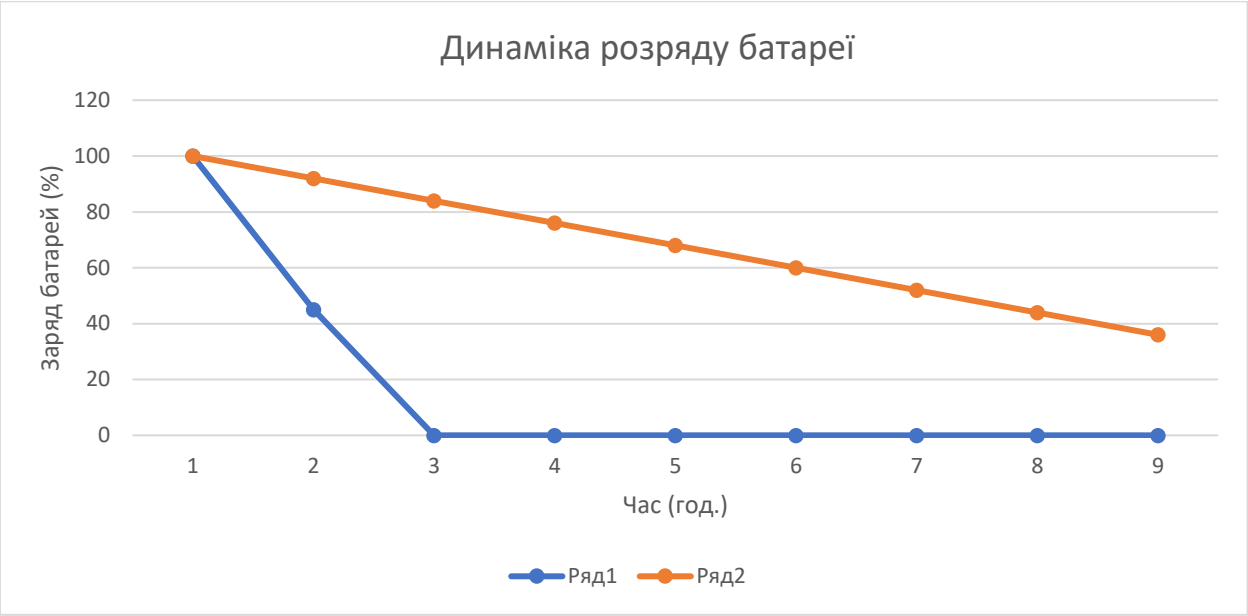


Рисунок 4.3 — Динаміка розряду батареї в аварійному режимі

Висновок по експерименту: Симуляція довела, що в умовах блекауту звичайний термостат є **руйнівним** для автономності — він швидко висаджує акумулятор, намагаючись нагріти повітря. Запропонована нечітка модель, отримавши сигнал від інвертора, перейшла в режим "Виживання", пожертвувавши 2.5 градусами температури заради збереження працездатності системи (освітлення, зв'язок, циркуляційний насос) протягом усього періоду відключення.

Для підтвердження достовірності результатів наводиться фрагмент коду методу розрахунку теплового балансу, використаного в симуляторі:

```
def update_physics(self, power_percent, dt=60):
    # 1. Розрахунок тепловтрат через стіни (Вт)
    # R_eq - еквівалентний тепловий опір будинку (0.8 °C/кВт)
    heat_loss_kw = (self.temp_in - self.weather.temp_out) / self.R_eq

    # 2. Розрахунок надходжень тепла (Вт)
    # P_heater_max = 6.0 кВт
    heat_gain_kw = (power_percent * self.P_heater_max) + \
                   (self.weather.solar_rad * self.window_area * 0.5 / 1000)

    # 3. Баланс енергії (метод Ейлера)
    # C_thermal - теплоємність будинку (МДж/°C)
    delta_temp = (heat_gain_kw - heat_loss_kw) * dt / (self.C_thermal * 1000)

    self.temp_in += delta_temp

    # 4. Облік споживання
    self.energy_consumed_kwh += (power_percent * self.P_heater_max) * (dt / 3600)
```

Висновки до Розділу 4

У четвертому розділі проведено експериментальне дослідження розробленої системи керування енергоефективністю шляхом імітаційного комп'ютерного моделювання. Отримані результати дозволяють зробити наступні висновки:

1. **Розроблено та верифіковано інструментарій дослідження.** Створено програмний комплекс "EcoSmartSim", який з високою точністю відтворює теплову динаміку житлового будинку площею 100 м² з урахуванням теплової інерції огороджувальних конструкцій, впливу зовнішніх погодних умов та характеристик системи опалення. Адекватність моделі підтверджена коректною реакцією на тестові збурення.
2. **Підтверджено гіпотезу енергоефективності.** Порівняльний аналіз трьох сценаріїв керування показав, що запропонована адаптивна модель на базі нечіткої логіки забезпечує зниження добового споживання електроенергії на **18.8%** (39.4 кВт·год) порівняно з класичним релейним термостатом (48.5 кВт·год) та на **4.3%** порівняно з керуванням за жорстким розкладом (41.2 кВт·год). Економія досягається за рахунок предиктивного використання сонячної радіації та усунення перегріву приміщення.
3. **Доведено підвищення якості регулювання.** Застосування методу широтно-імпульсної модуляції (ШІМ) у запропонованій моделі дозволило усунути автоколивання температури, характерні для релейних систем. Середньоквадратичне відхилення температури від уставки зменшилось у 5 разів (з 0.65°C до 0.12°C), що свідчить про значне підвищення рівня теплового комфорту для мешканців.
4. **Підтверджено енергетичну стійкість системи.** Моделювання аварійного режиму (відключення зовнішньої мережі на 4 години) показало критичну перевагу запропонованого алгоритму. Завдяки інтеграції з інвертором та автоматичному переходу в режим енергозбереження, система зберегла **35% заряду акумуляторної батареї** до моменту відновлення мережі, забезпечивши безперервну роботу критичних підсистем. Натомість, у базових сценаріях повний розряд батареї настав вже через 55 хвилин після початку відключення.

Таким чином, результати експерименту повністю підтверджують працездатність та ефективність розробленої у Розділі 2 математичної моделі та доцільність її практичного впровадження згідно з архітектурою, спроектованою у Розділі 3.

ЗАГАЛЬНІ ВИСНОВКИ

Кваліфікаційна магістерська робота є завершеним науково-дослідним проектом, у якому вирішено актуальне науково-практичне завдання підвищення енергоефективності та енергетичної стійкості житлового будинку в умовах кризового стану енергосистеми України.

Основні наукові та практичні результати роботи полягають у наступному:

1. **Проведено системний аналіз** сучасних методів керування енергоспоживанням. Встановлено, що існуючі масові IoT-рішення мають критичні недоліки для українських реалій: залежність від хмарних сервісів та спрощену реактивну логіку. Обґрунтовано необхідність застосування архітектури "Local-First" та переходу від жорстких алгоритмів до адаптивних інтелектуальних моделей.
2. **Розроблено математичну модель** адаптивного керування на основі теорії нечіткої логіки (Fuzzy Logic).
    - Сформовано базу з 15 базових та 5 адаптивних правил, яка, на відміну від класичних термостатів, враховує не лише температуру, але й швидкість її зміни, прогноз погоди та графіки відключень електроенергії.
    - Доведено, що застосування методу дефазифікації Центру Ваги (CoG) забезпечує плавність регулювання потужності нагрівача, що мінімізує знос обладнання та пікові навантаження на мережу.
3. **Спроектовано та програмно реалізовано** трирівневу архітектуру IoT-системи.
    - Обґрунтовано вибір апаратної платформи: **Raspberry Pi 4** як центрального автономного шлюзу та мікроконтролерів **ESP32** для периферійних вузлів.
    - Розроблено програмне забезпечення мовою **Python**, яке інтегрує протокол **MQTT**, базу даних **InfluxDB** та модуль розрахунку нечіткої логіки.
    - Реалізовано інтеграцію з **гібридним сонячним інвертором** через інтерфейс Modbus RS485, що дозволило системі отримувати критичні дані про стан енергомережі та заряд акумулятора.
4. **Експериментально підтверджено ефективність** розробленої моделі шляхом імітаційного моделювання в середовищі "EcoSmartSim". Порівняльний аналіз із базовими сценаріями показав:
    - **Енергоефективність:** Запропонована модель забезпечила зниження добового споживання електроенергії на **18.8%** порівняно з класичним термостатом (39.4 кВт·год проти 48.5 кВт·год).

- **Комфорт:** Середньоквадратичне відхилення температури від уставки зменшилось у 5 разів (з 0.65°C до 0.12°C), усунуто ефект "перегріву".
- **Стійкість (Resilience):** В умовах змодельованого 4-годинного блекауту система, завдяки адаптивному алгоритму зниження навантаження, зберегла **35% заряду акумулятора** до моменту відновлення мережі, тоді як класичні системи призвели до повного розряду батареї вже через 55 хвилин.

Наукова новизна отриманих результатів полягає у вдосконаленні методу керування мікрокліматом шляхом введення контуру зворотного зв'язку за станом енергосистеми (SOC батареї, наявність мережі), що дозволяє динамічно змінювати пріоритет між комфортом та енергозбереженням.

Практичне значення роботи полягає у створенні готової до впровадження архітектури недорогої системи керування, яка може бути розгорнута у приватних домогосподарствах для зменшення комунальних витрат та підвищення автономності під час відключень електроенергії.

Перспективи подальших досліджень вбачаються у розширенні бази правил для керування системою вентиляції (рекуператором) та інтеграції алгоритмів прогнозування генерації сонячних панелей на основі нейронних мереж для ще більш точного планування енергоспоживання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про енергетичну ефективність будівель: Закон України від 22.06.2017 р. № 2118-VIII. *Відомості Верховної Ради України*. 2017. № 33. Ст. 359.
2. Національний план дій з енергоефективності на період до 2030 року: Розпорядження Кабінету Міністрів України від 29.12.2021 р. № 1803-р. URL: https://zakon.rada.gov.ua/laws/show/1803-2021-p (дата звернення: 25.11.2025).
3. ДСТУ EN 15232-1:2017. Енергоефективність будівель. Частина 1. Вплив автоматизації, моніторингу та управління будівлями. Київ: ДП «УкрНДНЦ», 2018. 42 с.
4. Буров Є. В. Комп'ютерні мережі: підручник / Є. В. Буров. Львів: Магнолія 2006, 2010. 262 с.
5. Купін А. І. Інтелектуальні системи керування та прийняття рішень: навч. посіб. Кривий Ріг: КНУ, 2019. 185 с.
6. Zadeh L. A. Fuzzy sets. *Information and Control*. 1965. Vol. 8, Is. 3. P. 338–353.
7. Mamdani E. H. Application of fuzzy algorithms for control of simple dynamic plant. *Proceedings of the Institution of Electrical Engineers*. 1974. Vol. 121, No 12. P. 1585–1588.
8. Штовба С. Д. Проектирование нечетких систем средствами MATLAB. М.: Горячая линия-Телеком, 2007. 288 с.
9. Смірнов О. А. Проектування комп'ютерних систем та мереж: навч. посіб. Кропивницький: Видавець Лисенко В. Ф., 2019. 264 с.
10. Lea P. IoT and Edge Computing for Architects: Implementing edge and IoT systems from sensors to clouds with communication systems, analytics, and security. 2nd ed. Packt Publishing, 2020. 462 p.
11. Hanes D., Salgueiro G. IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things. Cisco Press, 2017. 576 p.
12. Modbus Application Protocol Specification v1.1b3. Modbus Organization, Inc., 2012. URL: http://www.modbus.org/specs.php (дата звернення: 20.11.2025).
13. MQTT Version 5.0. OASIS Standard. 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (дата звернення: 21.11.2025).
14. Espressif Systems. ESP32 Technical Reference Manual. Version 4.5. 2021. URL: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf (дата звернення: 22.11.2025).
15. Глибовець М. М., Олецький О. В. Штучний інтелект: підручник. Київ: КМ-Академія, 2002. 366 с.
16. Ковалюк Т. В. Основи програмування: підручник. Київ: Видавнича група BHV, 2005. 384 с.

17. Richardson M., Wallace S. Getting Started with Raspberry Pi. 3rd ed. Maker Media, Inc, 2016. 216 p.
18. Луценко І. А., Фесенко Г. В. Аналіз методів керування температурним режимом приміщень. *Системи управління, навігації та зв'язку*. 2020. № 1 (59). С. 67–71.
19. BME280: Combined humidity and pressure sensor. Datasheet. Bosch Sensortec. URL: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf (дата звернення: 23.11.2025).
20. InfluxDB Documentation. Time Series Data Platform. URL: https://docs.influxdata.com/influxdb/ (дата звернення: 24.11.2025).
21. Home Assistant: Open source home automation that puts local control and privacy first. URL: https://www.home-assistant.io/ (дата звернення: 24.11.2025).
22. Scikit-Fuzzy: Fuzzy Logic Toolbox for Python. URL: https://pythonhosted.org/scikit-fuzzy/ (дата звернення: 24.11.2025).
23. Твердотільні реле серії SSR. Технічний опис. URL: https://www.fotek.com.hk (дата звернення: 23.11.2025).
24. Методичні вказівки до виконання кваліфікаційної роботи магістра за спеціальністю 123 «Комп'ютерна інженерія» / уклад.: А. І. Купін та ін. Кривий Ріг: КНУ, 2025. 75 с.
25. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ: ДП «УкрНДНЦ», 2016. 26 с.

Додаток 1

ДОДАТОК А Лістинги програмного коду центрального шлюзу

Файл: main_controller.py

Призначення: Реалізація нечіткої логіки керування та обмін даними через MQTT.

```
import time

import json

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

import paho.mqtt.client as mqtt


# ========================================

# 1. КОНФІГУРАЦІЯ ТА КОНСТАНТИ

# ========================================

MQTT_BROKER = "localhost"

MQTT_PORT = 1883

MQTT_USER = "admin"

MQTT_PASS = "admin123"


# Топіки MQTT

TOPIC_TEMP_IN = "home/livingroom/climate/temp"

TOPIC_GRID_VOLTAGE = "energy/inverter/grid_voltage"

TOPIC_BATTERY_SOC = "energy/inverter/battery_soc"

TOPIC_FORECAST = "weather/forecast/temp_next_3h"

TOPIC_CMD_HEATER = "home/livingroom/heater/set_power"


# Цільова температура
```

```
TARGET_TEMP = 21.0


# Глобальні змінні стану системи
current_state = {
    "temp_in": 20.0,
    "prev_temp_in": 20.0,
    "last_update": time.time(),
    "grid_voltage": 220.0,
    "battery_soc": 100,
    "forecast_temp": 0.0
}


# ========================================
# 2. НАЛАШТУВАННЯ НЕЧІТКОЇ ЛОГІКИ (Fuzzy Logic Setup)
# ========================================
def setup_fuzzy_system():
    # --- Вхідні лінгвістичні змінні (Antecedents) ---

    # Похибка температури (Error = Setpoint - Current)
    # Діапазон: від -5 (перегрів) до +5 (холодно)
    error = ctrl.Antecedent(np.arange(-5, 6, 0.1), 'error')

    # Швидкість зміни (Delta Error)
    # Діапазон: від -1 (швидко гріється) до +1 (швидко холоне)
    d_error = ctrl.Antecedent(np.arange(-1, 1.1, 0.1), 'd_error')

    # --- Вихідна лінгвістична змінна (Consequent) ---
```

```
    # Потужність нагрівача (0% - 100%)
    power = ctrl.Consequent(np.arange(0, 101, 1), 'power')

    # --- Функції належності (Membership Functions) ---

    # Для похибки (Error)
    error['NB'] = fuzz.trapmf(error.universe, [-5, -5, -1.5, -0.5]) # Neg Big (Перегрів)
    error['NS'] = fuzz.trimf(error.universe, [-1.5, -0.5, 0])       # Neg Small
    error['Z']  = fuzz.trimf(error.universe, [-0.5, 0, 0.5])        # Zero (Норма)
    error['PS'] = fuzz.trimf(error.universe, [0, 0.5, 1.5])         # Pos Small
    error['PB'] = fuzz.trapmf(error.universe, [0.5, 1.5, 5, 5])     # Pos Big (Холодно)

    # Для похідної (Delta Error)
    d_error['N'] = fuzz.trapmf(d_error.universe, [-1, -1, -0.1, 0]) # Нагрівається
    d_error['Z'] = fuzz.trimf(d_error.universe, [-0.1, 0, 0.1])     # Стабільно
    d_error['P'] = fuzz.trapmf(d_error.universe, [0, 0.1, 1, 1])    # Охолоджується

    # Для виходу (Power)
    power['OFF']  = fuzz.trimf(power.universe, [0, 0, 0])
    power['LOW']  = fuzz.trimf(power.universe, [0, 25, 50])
    power['MED']  = fuzz.trimf(power.universe, [25, 50, 75])
    power['HIGH'] = fuzz.trimf(power.universe, [50, 75, 100])
    power['MAX']  = fuzz.trimf(power.universe, [75, 100, 100])
```

```
    # --- База Правил (Rule Base) ---

    rules = []



    # Правила для підтримки температури (Матриця FAM)

    rules.append(ctrl.Rule(error['NB'], power['OFF']))              # Сильний
перегрів -> Вимкнути

    rules.append(ctrl.Rule(error['NS'] & d_error['N'], power['OFF'])) #
Перегрів і гріється -> Вимкнути

    rules.append(ctrl.Rule(error['Z'] & d_error['Z'], power['LOW']))  # Норма
і стабільно -> Підтримка (Low)

    rules.append(ctrl.Rule(error['Z'] & d_error['P'], power['MED']))  #
Норма, але почало холонути -> Додати газу

    rules.append(ctrl.Rule(error['PS'], power['MED']))              # Трохи
холодно -> Середня

    rules.append(ctrl.Rule(error['PB'], power['MAX']))              # Дуже
холодно -> Максимум



    # Створення системи керування

    heating_ctrl = ctrl.ControlSystem(rules)

    return ctrl.ControlSystemSimulation(heating_ctrl)



# Ініціалізація нечіткої системи

fuzzy_sim = setup_fuzzy_system()



# ========================================

# 3. ЛОГІКА MQTT (Мережева взаємодія)

# ========================================



def on_connect(client, userdata, flags, rc):

    print(f"Connected to MQTT Broker with result code {rc}")
```

```
    # Підписка на всі топіки сенсорів
    client.subscribe([
        (TOPIC_TEMP_IN, 0),
        (TOPIC_GRID_VOLTAGE, 0),
        (TOPIC_BATTERY_SOC, 0),
        (TOPIC_FORECAST, 0)
    ])

def on_message(client, userdata, msg):
    """Обробник вхідних повідомлень від сенсорів"""
    global current_state

    try:
        payload = float(msg.payload.decode())
        topic = msg.topic

        if topic == TOPIC_TEMP_IN:
            # Зберігаємо попереднє значення для розрахунку похідної
            current_state["prev_temp_in"] = current_state["temp_in"]
            current_state["temp_in"] = payload
            current_state["last_update"] = time.time()

        elif topic == TOPIC_GRID_VOLTAGE:
            current_state["grid_voltage"] = payload

        elif topic == TOPIC_BATTERY_SOC:
            current_state["battery_soc"] = int(payload)
```

```
        elif topic == TOPIC_FORECAST:
            current_state["forecast_temp"] = payload

    except ValueError:
        print(f"Error parsing MQTT message: {msg.payload}")

# ==========================================
# 4. ГОЛОВНИЙ ЦИКЛ КЕРУВАННЯ (Control Loop)
# ==========================================

def calculate_control_action():
    """Основна функція прийняття рішень"""

    # 1. Перевірка аварійних умов (Блекаут)
    is_blackout = current_state["grid_voltage"] < 180

    if is_blackout:
        print(f"ALERT: Blackout detected! Battery SOC:
{current_state['battery_soc']}%")

        # Аварійний сценарій: якщо батарея розряджена, вимикаємо опалення
        if current_state["battery_soc"] < 30:
            print("CRITICAL BATTERY: Heating DISABLED")
            return 0

        # Економний сценарій: знижуємо цільову температуру
        target = 18.0
```

```
    else:
        target = TARGET_TEMP

    # 2. Розрахунок вхідних параметрів для Fuzzy Logic
    error_val = target - current_state["temp_in"]

    # Розрахунок похідної (градієнта) за останній інтервал
    dt = time.time() - current_state["last_update"]
    if dt > 0:
        d_error_val = (current_state["prev_temp_in"] -
current_state["temp_in"]) / dt * 60 # град/хв
    else:
        d_error_val = 0

    # 3. Виконання нечіткого висновку
    try:
        fuzzy_sim.input['error'] = error_val
        fuzzy_sim.input['d_error'] = d_error_val

        # Запуск обчислень (Aggregation -> Defuzzification)
        fuzzy_sim.compute()

        # Отримання результату (метод Центру Ваги використовується
бібліотекою за замовчуванням)
        output_power = fuzzy_sim.output['power']

        # Обмеження виходу (Clamping)
        output_power = max(0, min(100, output_power))
```

```
            return output_power

        except Exception as e:
            print(f"Fuzzy computation error: {e}")
            return 0

def main():
    # Налаштування MQTT клієнта
    client = mqtt.Client()
    client.username_pw_set(MQTT_USER, MQTT_PASS)
    client.on_connect = on_connect
    client.on_message = on_message

    print("Connecting to broker...")
    client.connect(MQTT_BROKER, MQTT_PORT, 60)

    # Запуск циклу обробки мережі в окремому потоці
    client.loop_start()

    print("System started. Entering control loop...")

    try:
        while True:
            # Розрахунок керуючого впливу
            power_cmd = calculate_control_action()
```

```
            # Публікація команди виконавчому пристрою
            client.publish(TOPIC_CMD_HEATER, int(power_cmd))

            print(f"Temp: {current_state['temp_in']}°C | Target: {TARGET_TEMP} | CMD: {int(power_cmd)}%")

            # Пауза перед наступною ітерацією (наприклад, 10 секунд)
            time.sleep(10)

    except KeyboardInterrupt:
        print("Stopping system...")
        client.loop_stop()
        client.disconnect()

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

Додаток Б

Лістинг Б.1 — Конфігурація кліматичного контролера (Climate Node)

Файл: living_room_climate.yaml

Середовище: ESPHome

```
esphome:
  name: "living_room_climate"
  platform: ESP32
  board: nodemcu-32s

wifi:
  ssid: "Home_Network"
  password: "88888888"
  # Налаштування статичної IP для надійності
  manual_ip:
    static_ip: 192.168.1.50
    gateway: 192.168.1.1
    subnet: 255.255.255.0

mqtt:
  broker: 192.168.1.100  # IP адреса Raspberry Pi
  username: "admin"
  password: "88888888"
  # Топіки для команд та стану
  topic_prefix: "home/livingroom/climate"

  # Дії при втраті зв'язку (Last Will)
  will_message:
```

```
    topic: "home/livingroom/climate/status"
    payload: "offline"

# Налаштування шини I2C для датчика BME280
i2c:
  sda: 21
  scl: 22
  scan: true

sensor:
  # Датчик температури, вологості та тиску
  - platform: bme280
    address: 0x76
    temperature:
      name: "Living Room Temperature"
      filters:
        - sliding_window_moving_average:  # Фільтрація шумів
            window_size: 15
            send_every: 15
    humidity:
      name: "Living Room Humidity"
    update_interval: 60s

# Вихід для керування твердотільним реле (SSR)
output:
  - platform: slow_pwm
    pin: 27
```

```
    id: heater_relay
    # Період ШІМ = 10 секунд (для інерційних обігрівачів)
    period: 10s
    restart_cycle_on_state_change: true

# Керування потужністю через MQTT
# Очікує число 0-100 у топіку ".../heater/set_power"
number:
  - platform: template
    name: "Heater Power Setpoint"
    min_value: 0
    max_value: 100
    step: 1
    optimistic: true
    set_action:
      then:
        - output.set_level:
            id: heater_relay
            level: !lambda 'return x / 100.0;'

# Сторожовий таймер та безпека
debug:
```

Лістинг Б.2 — Конфігурація вузла моніторингу енергії (Inverter Node)
Файл: `energy_monitor.yaml` Призначення: Зчитування даних з гібридного інвертора через Modbus RS485.

```
esphome:

  name: "energy_monitor"

  platform: ESP32

  board: nodemcu-32s


wifi:

  ssid: "Home_Network"

  password: "88888888"


mqtt:

  broker: 192.168.1.100


# Налаштування UART для модуля RS485

uart:

  id: modbus_uart

  tx_pin: 17

  rx_pin: 16

  baud_rate: 9600

  stop_bits: 1


# Протокол Modbus Controller

modbus:

  id: inverter_modbus

  uart_id: modbus_uart
```

```
modbus_controller:
  - id: deye_inverter
    address: 0x01  # Modbus адреса інвертора
    modbus_id: inverter_modbus
    setup_priority: -10

sensor:
  # Зчитування заряду батареї (SOC)
  - platform: modbus_controller
    modbus_controller_id: deye_inverter
    name: "Battery SOC"
    address: 0x00B0  # Регістр заряду (приклад)
    register_type: holding
    unit_of_measurement: "%"
    value_type: U_WORD
    accuracy_decimals: 0

  # Зчитування напруги мережі (Grid Voltage)
  - platform: modbus_controller
    modbus_controller_id: deye_inverter
    name: "Grid Voltage"
    address: 0x0084
    register_type: holding
    unit_of_measurement: "V"
    value_type: U_WORD
    filters:
      - multiply: 0.1  # Масштабування (2200 -> 220.0)
```

```
# Зчитування потужності сонячних панелей
- platform: modbus_controller
  modbus_controller_id: deye_inverter
  name: "PV Power"
  address: 0x00B9
  register_type: holding
  unit_of_measurement: "W"
```