Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій

Бакалаврська кваліфікаційна робота на тему:

«АДАПТИВНА СИСТЕМА УПРАВЛІННЯ ОПАЛЕННЯМ БУДИНКУ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ»

Виконав: студент 4 курсу, групи 2ІСТ-21б
спеціальності 126 – «Інформаційні
системи та технології»
_____ Вадим НАГОРНЯК
Керівник: к.т.н., доц. каф. САІТ
_____ Георгій ГОРЯЧЕВ
«03» 06 2025 р.
Рецензент: к.ф.-м.н., доц. каф. КН
_____ Олександр ШЕВЧУК
«10» 06 2025 р.

Допущено до захисту

Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
«06» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій
Рівень вищої освіти – перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність 126 – «Інформаційні системи та технології»
Освітньо-професійна програма – Прикладні інформаційні технології

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
д.т.н., проф. Віталій МОКІН
"28" 05 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Нагорняку Вадиму Євгеновичу

1. Тема роботи: «Адаптивна система управління опаленням будинку з використанням штучного інтелекту»
керівник роботи: Горячев Г. В., к.т.н., доц. каф. САІТ
затверджені наказом ВНТУ від «20» 05 2025 року № 97
2. Термін подання студентом роботи 31.05.2025 р.
3. Вихідні дані до роботи:
1) Дані з сенсорів температури, вологості, рівня CO_2 та присутності в приміщенні.
2) Поведінкові шаблони користувачів та сценарії взаємодії з системою.
4. Зміст текстової частини:
1) Аналіз існуючих систем адаптивного опалення.
2) Вибір технологій та архітектури для реалізації системи.
3) Розроблення адаптивної системи керування опаленням з використанням штучного інтелекту.
5. Перелік ілюстративного матеріалу:
1) Архітектура адаптивної системи опалення на базі Home Assistant;
2) Use Case діаграма системи опалення;
3) Діаграма класів системи керування опаленням з елементами ШІ;
4) Діаграма послідовностей для обробки MQTT-сценарію прийняття рішення;
5) Діаграма діяльності логіки взаємодії з моделлю Gemini.

6. Консультанти розділів роботи:

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

7. Дата видачі завдання 28.03.25 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз існуючих систем адаптивного опалення	*01.04.2025*	*20.04.2025*	
2	Вибір відкритих платформ і методів інтелектуального управління	*20.04.2025*	*30.04.2025*	
3	Розроблення структури системи та інтеграція з Home Assistant	*01.05.2025*	*10.05.2025*	
4	Реалізація інтелектуального модуля з інтеграцією Gemini API	*10.05.2025*	*20.05.2025*	
5	Оформлення матеріалів до захисту БКР	*20.05.2025*	*30.05.2025*	

Студент _____ Вадим НАГОРНЯК

Керівник роботи _____ Георгій ГОРЯЧЕВ

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 73 сторінок формату А4, містить 27 рисунків, список використаних джерел налічує 21 найменувань.

Метою роботи є розроблення адаптивної системи управління опаленням будинку з використанням елементів штучного інтелекту.

У роботі проведено аналіз існуючих рішень у сфері смарт-опалення, обґрунтовано доцільність створення власної системи. Реалізовано архітектуру на базі платформи Home Assistant, створено інтелектуальний модуль прийняття рішень на мові Python із використанням API моделі Google Gemini. Запропонована система демонструє адаптивність, енергоефективність і можливість масштабування.

Ключові слова: адаптивне опалення, штучний інтелект, Home Assistant, Python, Google Gemini API, інтелектуальний модуль.

ABSTRACT

The bachelor's thesis consists of 73 A4 pages, including 27 figures, and a list of 21 references.

The aim of the work is to develop an adaptive home heating control system using elements of artificial intelligence.

The paper analyzes existing smart heating solutions and justifies the feasibility of creating a custom system. The architecture is implemented using the Home Assistant platform, and an intelligent decision-making module was developed in Python with integration of the Google Gemini API. The proposed system demonstrates adaptability, energy efficiency, and scalability potential.

Key words: adaptive heating, artificial intelligence, Home Assistant, Python, Google Gemini API, intelligent module.

ЗМІСТ

ВСТУП 4
1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПОСТАНОВКА ЗАДАЧІ 6
1.1. Огляд сучасних комерційних систем адаптивного опалення (Nest, Tado, Honeywell) 6
1.2. Можливості відкритих платформ (Home Assistant, OpenHAB) щодо інтеграції ШІ 9
1.3 Порівняння методів керування: класичні та інтелектуальні підходи 14
1.4 Формулювання проблеми, мета дослідження, критерії ефективності, постановка задачі 16
1.5 Обґрунтування доцільності створення власної системи 18
1.6 Висновки 20
2 РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ АДАПТИВНОГО ОПАЛЕННЯ 22
2.1 Архітектура системи: компоненти, принципи взаємодії, середовище виконання 22
2.2 Моделювання архітектури адаптивної системи засобами UML 24
2.3 Вибір оптимальної IT-інфраструктури 28
2.4 Вибір апаратного та програмного забезпечення, інтеграція з Home Assistant 30
2.5 Реалізація автоматизацій, логіка прийняття рішень, взаємодія з інтелектуальним модулем 33
2.6 Приклади реалізованих сценаріїв автоматизації та візуалізація в Home Assistant 34
2.7 Тестування системи, результати та аналіз ефективності 36
2.8 Використані інструменти: практичні аспекти реалізації 37
2.9 Висновки 42
3 ТЕСТУВАННЯ, ОЦІНЮВАННЯ РЕЗУЛЬТАТІВ ТА ВИСНОВКИ 43
3.1 Методика тестування: сценарний підхід на основі моделювання 43
3.2 Аналіз ефективності: економія ресурсів, стабільність роботи системи 47
3.3 Інтелектуальний аналіз даних та підтримка прийняття рішень 49
3.4 Оцінювання комфортності системи та критерії ефективності 51
3.5 Висновки 54
ВИСНОВКИ 56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 58

Додаток А (обов’язковий) Технічне завдання 60
Додаток Б (обов’язковий) Протокол перевірки кваліфікаційної роботи 62
Додаток В (довідниковий) Фрагмент лістингу програми 63
Додаток Г (обов’язковий) Ілюстративна частина 70

ВСТУП

Актуальність теми. Сучасні тенденції енергоефективності та автоматизації побутових процесів зумовлюють зростання інтересу до розумних систем управління кліматом у житлових приміщеннях. З огляду на зростання вартості енергоносіїв і потребу в комфортних умовах проживання, особливої актуальності набуває впровадження адаптивних систем опалення, здатних самостійно приймати рішення на основі змін зовнішніх та внутрішніх факторів. Інтеграція елементів штучного інтелекту у такі системи відкриває нові можливості для точного керування, зменшення витрат на опалення та підвищення рівня автоматизації. Зокрема, використання мовних моделей нового покоління (наприклад, Google Gemini) дозволяє формувати більш гнучкі й контекстно-залежні сценарії поведінки системи, що відповідає сучасним вимогам до “розумного будинку”.

Мета і задачі дослідження. Метою дослідження є розроблення адаптивної інформаційної системи управління опаленням будинку з використанням елементів штучного інтелекту.

Для досягнення мети необхідно вирішити такі задачі:

- провести аналіз сучасних комерційних і відкритих рішень у сфері смарт-опалення;
- обґрунтувати доцільність створення власної адаптивної системи;
- спроєктувати архітектуру на базі платформи Home Assistant;
- реалізувати модуль прийняття рішень із використанням Python і Google Gemini API;
- розробити сценарії автоматизації з урахуванням прогнозу та поточних умов;
- протестувати систему й оцінити її ефективність.

Об’єкт дослідження — процес створення адаптивної системи управління опаленням у житловому будинку.

Предмет дослідження — методи та програмні засоби для реалізації

інтелектуального керування кліматичними умовами приміщення на основі аналізу даних.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Огляд сучасних комерційних систем адаптивного опалення (Nest, Tado, Honeywell)

Nest — це система, яку розробила компанія, що входить до складу Google. Основна ідея — адаптація до поведінки користувача. Термостат може “вивчати” розпорядок дня та автоматично регулювати температуру без постійного ручного втручання. Є мобільний застосунок, підтримка голосових помічників і інтеграція з іншими пристроями Google (рис. 1.1). Працює через інтернет, дані зберігаються в хмарі [1].

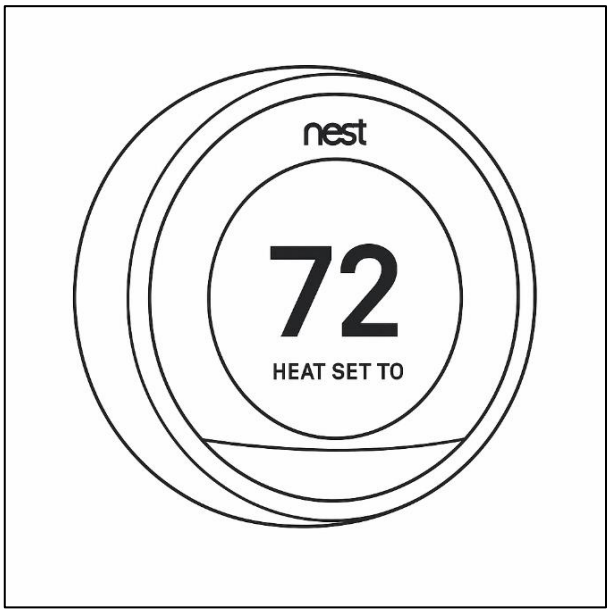


Рисунок 1.1 – Розумний термостат Nest (створено автором на основі зовнішнього вигляду оригінального пристрою)

Tado — ще одне відоме рішення (рис. 1.2). Тут акцент зроблений на геолокацію: система визначає, коли користувач вдома, і відповідно вмикає або вимикає опалення. Є можливість керувати температурою в кожній кімнаті окремо, якщо використовуються відповідні термоголовки. Частина функцій, таких як розширена аналітика або автоматичне виявлення відкритих вікон, доступна лише за підпискою.

Рисунок 1.2 – Розумний термостат Tado (створено автором на основі зовнішнього вигляду оригінального пристрою)

Honeywell — виробник із великим досвідом у галузі керування кліматом (рис. 1.3). Вони пропонують широку лінійку термостатів та систем автоматизації, які можна використовувати як у квартирах, так і в приватних будинках. Часто застосовуються в комплексних рішеннях разом із системами безпеки або вентиляції. Інтерфейс більш класичний, але функціональність гнучка, а налаштування детальні [1].

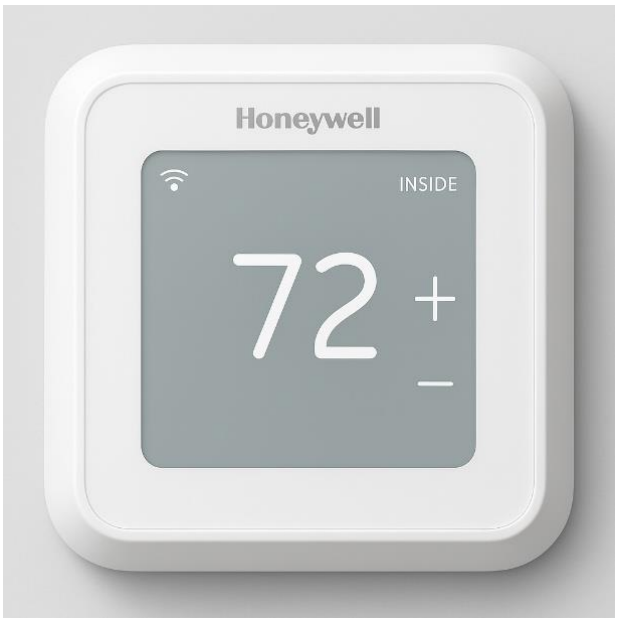


Рисунок 1.3 – Розумний термостат Honeywell (створено автором на основі зовнішнього вигляду оригінального пристрою)

На сучасному ринку систем адаптивного опалення найпоширенішими рішеннями залишаються такі продукти, як Google Nest, Tado та Honeywell. Вони орієнтовані переважно на побутове використання і розраховані на масового споживача з базовими потребами в автоматизації клімату. Водночас детальний аналіз їхніх можливостей показує, що ці системи мають низку обмежень, які можуть виявитися критичними для більш гнучких або нестандартних сценаріїв.

Так, система Nest підтримує автоматичне навчання поведінки користувача, проте не має підтримки кімнатного зонування, що обмежує її ефективність у багатокімнатних приміщеннях. Геолокаційне керування реалізоване лише частково і залежить від інтеграції з екосистемою Google. Важливо також, що повна функціональність Nest можлива лише при постійному підключенні до Інтернету. Хоча додаткові підписки не є обов'язковими, використання сторонніх пристроїв або платформ часто ускладнюється.

У випадку Tado спостерігається протилежна ситуація: система не навчається автоматично, але забезпечує повне кімнатне зонування, гнучке геолокаційне керування та сумісність із великою кількістю голосових помічників (Google Assistant, Alexa, Apple HomeKit). Однак Tado блокує частину ключового функціоналу за підпискою, зокрема автоматичні сценарії оптимізації, що є суттєвим недоліком при довготривалому використанні.

Honeywell, у свою чергу, має сильні позиції в застосуванні в рамках систем керування будівлями (BMS) і надає підтримку кімнатного зонування. Проте її інтелектуальні можливості є доволі обмеженими: навчання поведінки реалізовано частково, а геолокаційне керування часто потребує додаткових шлюзів або додатків. Сумісність з іншими платформами є широкою, але вимагає додаткової конфігурації, а локальна робота без інтернету — не завжди можлива.

Загальною проблемою всіх цих рішень є залежність від хмарних сервісів, відсутність прозорості в алгоритмах прийняття рішень, а також обмежені можливості кастомізації. Користувач обмежений запропонованими сценаріями і не має повного контролю над логікою роботи системи.

У цьому контексті створення власної системи на базі відкритої платформи

Home Assistant із підключенням мовної моделі (LLM), яка здатна аналізувати сенсорні дані і формувати текстові рекомендації, є цілком обґрунтованим. Такий підхід дозволяє не лише уникнути щомісячних підписок і комерційних обмежень, а й забезпечити повний контроль, адаптивність і локальну автономність роботи системи.

1.2. Можливості відкритих платформ (Home Assistant, OpenHAB) щодо інтеграції ШІ

У процесі вивчення теми адаптивного опалення я звернув увагу не лише на комерційні рішення, а й на відкриті платформи, які дозволяють повністю контролювати логіку автоматизації. Найвідоміші з них — Home Assistant та OpenHAB. Обидві системи орієнтовані на користувачів, які хочуть самостійно налаштувати свою систему розумного будинку під власні потреби, без обмежень, які зазвичай є у закритих рішеннях.

На відміну від комерційних продуктів, відкриті платформи надають можливість змінювати й розширювати функціонал за рахунок модулів, інтеграцій, скриптів та сторонніх інструментів. Їх можна розгорнути на локальному сервері або недорогому мінікомп’ютері, типу Raspberry Pi, і мати повний контроль над даними — без залежності від хмарних сервісів.

Особливо важливим є те, що обидві системи дозволяють інтегрувати елементи штучного інтелекту (рис. 1.4): наприклад, зв’язуватися з моделями машинного навчання, відправляти дані на зовнішній сервер для обробки, або використовувати вже готові інструменти прогнозування. Усе це дає змогу не просто керувати пристроями за заданим розкладом, а приймати рішення на основі реальних даних — наприклад, враховуючи звички користувача, температуру на вулиці, час доби чи рівень вуглекислого газу в кімнаті.

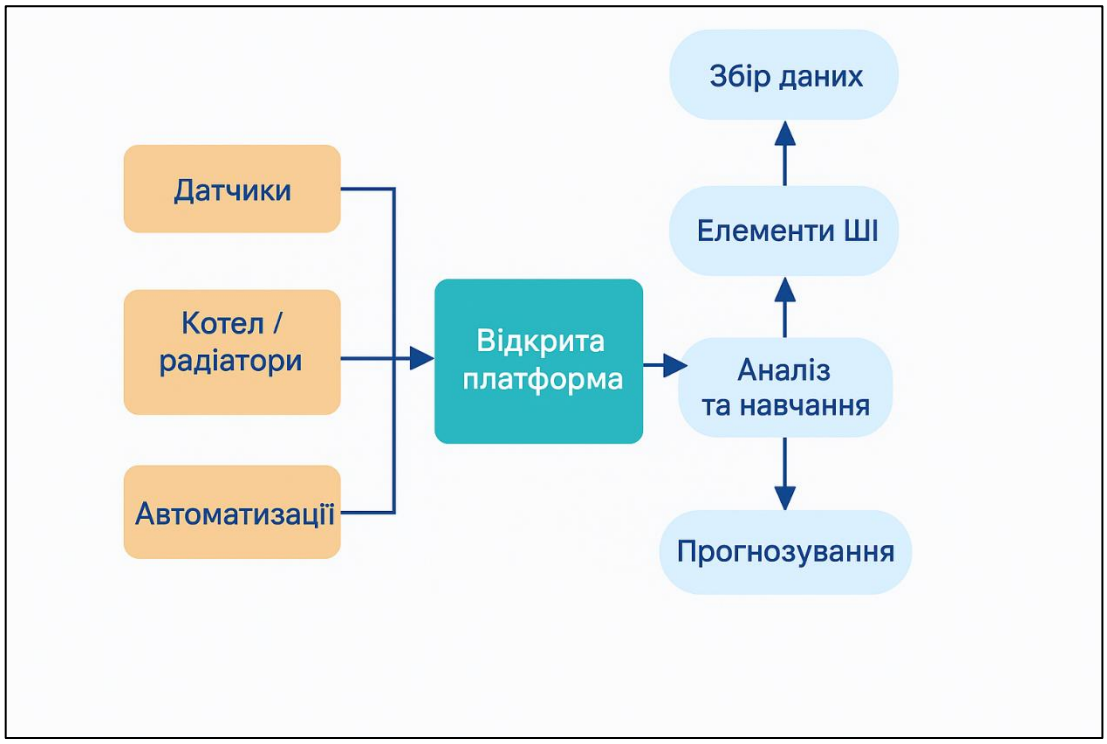


Рисунок 1.4 – Схема взаємодії відкритої платформи з компонентами системи та елементами ШІ

Home Assistant — це відкрита платформа для автоматизації розумного будинку, яка активно розвивається спільнотою і має широку підтримку пристроїв. Що мені сподобалося — вона встановлюється локально, наприклад, на Raspberry Pi або мінісервер, і не залежить від постійного підключення до хмари. Це не тільки дає більше контролю над даними, а й дозволяє будувати більш гнучку логіку автоматизації.

У Home Assistant інтеграції реалізуються через YAML-конфігурації або UI, і підтримується велика кількість протоколів: MQTT, Zigbee, Z-Wave, REST API та інші. Завдяки цьому до платформи можна підключити практично будь-який сенсор, розумну розетку, термоголовку, камеру чи навіть голосового помічника [2,3].

Щодо використання ШІ, можливості доволі широкі. Платформа не має “вбудованого” штучного інтелекту, але дозволяє створювати взаємодію з ML-моделями через зовнішні скрипти. Наприклад, можна написати Python-скрипт, який отримує дані з Home Assistant (температуру, присутність, історію використання), обробляє їх за допомогою моделі, а потім повертає прогноз або

керуючу команду назад у систему. Такий підхід дозволяє будувати адаптивну поведінку — не просто “включи/вимкни”, а “проаналізуй ситуацію і прийми оптимальне рішення”.

Також Home Assistant підтримує готові інтеграції з голосовими асистентами, такими як Google Assistant та Amazon Alexa, що значно спрощує взаємодію з системою шляхом голосового керування. Це дозволяє користувачу здійснювати перемикання режимів, отримувати зворотний зв’язок або керувати сценаріями безпосередньо за допомогою голосових команд. Крім того, через REST API платформа відкриває широкі можливості для підключення сторонніх сервісів, зокрема таких як OpenAI, TensorFlow або навіть локальних нейромереж, які були самостійно навчені для конкретного середовища. Це забезпечує не лише розширення функціональності, а й дає змогу реалізувати більш складну логіку — прогнозування поведінки мешканців, адаптацію до ритму життя, оптимізацію енергоспоживання на основі історичних даних. Саме така відкритість до інтеграцій та можливість створення справжньої інтелектуальної системи (рис. 1.5) здалися мені особливо цінними й перспективними у контексті реалізації власного проєкту.

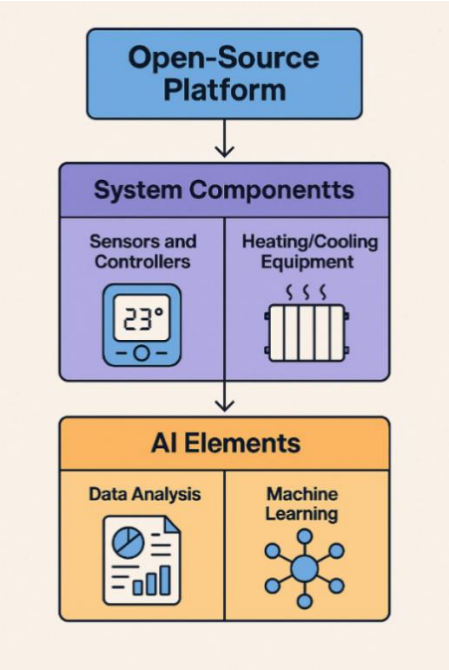


Рисунок 1.5 – Взаємодія Home Assistant з зовнішньою моделлю машинного навчання

OpenHAB — ще одна популярна відкрита система автоматизації, яка, на відміну від Home Assistant, написана на Java. Це робить її кросплатформенною й стабільною на різних типах пристроїв — можна запускати як на звичайному ПК, так і на Linux-сервері, або тому ж Raspberry Pi.

Структура в OpenHAB побудована навколо понять “Things”, “Items”, “Rules” і “Sitemaps”. Спочатку я трохи плутався в цій термінології, але з часом стало зрозуміло: "Thing" — це реальний пристрій (наприклад, термостат), "Item" — логічний представник цього пристрою в системі, а правила (Rules) — це вже автоматизація, яка керує діями. Такий підхід надає велику гнучкість у побудові системи.

OpenHAB має дуже потужні можливості для інтеграції, особливо якщо потрібно підключити щось нестандартне. Наприклад, можна використовувати Jython — версію Python для Java, яка дозволяє писати скрипти для складної автоматизації. Це відкриває двері для реалізації сценаріїв на основі аналітики чи навіть машинного навчання.

Із цікавого — OpenHAB можна зв’язати з зовнішнім ML-сервером або API, через який надсилати дані (наприклад, температуру, вологість, графік присутності), обробляти їх за допомогою моделі, й отримувати назад прогноз або команду. Це може бути як локальна модель, так і щось хмарне — головне, щоб був API або брокер MQTT [4].

Також підтримується інтеграція з такими популярними голосовими асистентами, як Google Assistant та Amazon Alexa, а також із різноманітними сервісами розпізнавання мовлення, що дає змогу реалізувати голосове керування опаленням, світлом, безпекою тощо. Це створює додаткову зручність для користувача, особливо в умовах hands-free взаємодії або для людей з обмеженими можливостями. Крім того, Home Assistant дозволяє зв’язуватися з камерами відеоспостереження і, при бажанні, спрямовувати потоки відео на зовнішні сервіси для обробки за допомогою технологій комп’ютерного зору. Наприклад, можна реалізувати виявлення присутності, розпізнавання обличчя (рис. 1.6) або визначення активності для більш точного керування кліматом.

Загалом, хоча платформа і є більш “низькорівневою” порівняно з комерційними рішеннями, її головною перевагою є саме гнучкість, можливість тонкої адаптації до будь-яких потреб користувача та висока стабільність у щоденному використанні.

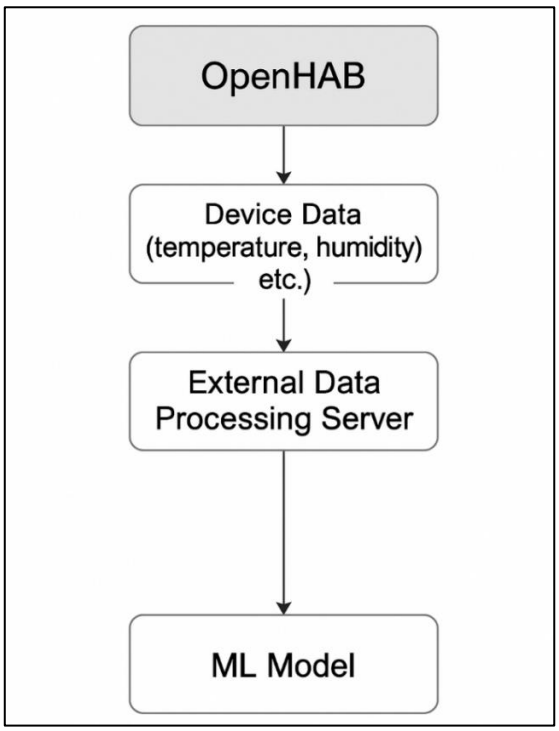


Рисунок 1.6 – Взаємодія OpenHAB із зовнішнім сервером обробки даних або ML-моделлю

Після ознайомлення з обома платформами я побачив, що кожна має свої сильні сторони, особливо якщо розглядати їх у зв’язці з використанням машинного навчання чи інших елементів штучного інтелекту.

Home Assistant простіший в освоєнні, має активну спільноту, зручний веб-інтерфейс і вже вбудовані інструменти для інтеграції з різноманітними API та хмарними сервісами. Важливо те, що інтеграції легко додавати, а автоматизації створюються у зручному форматі. Саме ця платформа, на мою думку, краще підходить для швидкого старту і експериментів із підключенням зовнішніх ML-скриптів, особливо через Python.

OpenHAB, у свою чергу, більше орієнтований на стабільну роботу та довготривалі рішення (табл. 1.1). Його архітектура гнучка, але трохи складніша

в налаштуванні. Для інтеграції з ШІ також є можливості, але вони більше орієнтовані на досвідчених користувачів — через Jython або зовнішні API-з’єднання.

Таблиця 1.1 – Порівняння платформ Home Assistant і OpenHAB у контексті інтеграції ШІ

Критерій	Home Assistant	OpenHAB
Простота налаштування	Висока	Середня
Підтримка інтеграцій із ШІ	Пряма (через API)	Можлива (через API)
Підтримка Python/скриптів	Так (Python)	Так (Jython)
Зв’язок із зовнішніми ML-моделями	Так (через REST/MQTT)	Так (через API/MQTT)
Інтерфейс користувача	Зручний	Більш технічний
Активність спільноти	Дуже активна	Активна
Гнучкість автоматизації	Гнучка та зрозуміла	Дуже гнучка, але складна
Локальна обробка даних	Так	Так

1.3 Порівняння методів керування: класичні та інтелектуальні підходи

Коли я почав заглиблюватись у способи керування опаленням, перше, на що звернув увагу — це класичні ПІД-регулятори. Це досить старий, але досі популярний метод, який базується на трьох компонентах: пропорційному, інтегральному та диференційному. Вони відповідають за те, як система реагує на помилки між бажаною та реальною температурою.

Перевага ПІД-регулятора в тому, що він простий, надійний і не потребує великих обчислювальних ресурсів. Але є і мінус — він не адаптується до змін умов. Тобто якщо погода раптово змінилася, або вдома стало більше людей — алгоритм не знає, як на це реагувати, окрім як “крутити” ту ж формулу.

Згодом я ознайомився з інтелектуальними підходами — зокрема, з використанням моделей машинного навчання. Ці методи дають змогу враховувати набагато більше факторів: зовнішню температуру, вологість, час доби, графік присутності мешканців, історію змін температури в приміщенні.

На основі цього модель може робити прогнози — наприклад, коли вмикати обігрів раніше, щоб до вечора кімната вже була теплою, або коли варто знизити температуру, бо скоро всі підуть з дому [5].

Особливо цікавою виявилась ідея предиктивного керування — коли система не просто реагує, а передбачає, що буде далі. Наприклад, якщо програма знає, що завтра очікується похолодання, вона може заздалегідь змінити режим опалення. Або якщо бачить, що користувач кожного дня повертається додому о 18:00, — почне гріти трохи раніше, щоб створити комфорт до його приходу [6,7].

Інтелектуальні підходи вимагають більше ресурсів і складнішої реалізації, але водночас вони значно гнучкіші й ефективніші. У порівнянні з ПІД-регуляторами — це зовсім інший рівень керування. Там, де класичні методи працюють “по факту”, інтелектуальні намагаються передбачити майбутнє.

У порівнянні різних методів керування — ПІД-регуляторів, моделей машинного навчання (ML) та предиктивних моделей — чітко простежуються ключові відмінності за низкою критеріїв.

ПІД-регулятори вирізняються низькою складністю реалізації та не потребують історичних даних, проте мають низьку адаптивність до змін і середню точність прогнозів. Вони також характеризуються низькими вимогами до обчислювальних ресурсів і не здатні до навчання чи прогнозування майбутніх станів.Моделі машинного навчання мають середню або високу складність реалізації. Їм необхідні історичні дані для навчання. Водночас вони забезпечують високу адаптивність і точність прогнозів, мають середні вимоги до ресурсів, можуть навчатися та, хоч і обмежено, але здібні до прогнозування майбутніх станів. Предиктивні моделі є найскладнішими у реалізації й потребують високоякісних та об’ємних даних. Вони демонструють високу адаптивність, дуже високу точність прогнозів і вимагають значних

обчислювальних потужностей. Ці моделі також здатні до навчання та ефективного прогнозування майбутніх станів системи.

Таким чином, вибір конкретного підходу залежить від цілей, доступних ресурсів і необхідного рівня інтелектуалізації системи.

1.4 Формулювання проблеми, мета дослідження, критерії ефективності, постановка задачі

Аналіз існуючих рішень у сфері керування побутовим опаленням показав, що більшість комерційних систем мають суттєві обмеження. Часто вони розроблені для масового користувача й орієнтовані на певні ринки, що знижує їхню гнучкість в умовах локальної адаптації. У таких системах обмежено можливості глибокого налаштування, а функціонал, який можна було б використати для гнучкої поведінки системи, часто відкривається лише у платній підписці або через хмарні сервіси, що не завжди є прийнятним варіантом. Водночас більшість доступних систем працюють на базі наперед визначених сценаріїв або розкладів, які не враховують поточний контекст і не реагують на поведінкові особливості мешканців.

Зі зростанням доступності відкритих платформ для автоматизації, таких як Home Assistant, з'являється можливість створення власних рішень, які можуть бути повністю адаптовані під потреби конкретного користувача, не вимагаючи підключення до закритих інфраструктур. Особливої актуальності набуває поєднання таких платформ із сучасними інтелектуальними сервісами, зокрема великими мовними моделями (LLM), які здатні аналізувати текстові запити, формувати рекомендації на основі опису ситуації та працювати з природною мовою. Такі інструменти дозволяють винести логіку прийняття рішень за межі жорстких умовних операторів, що характерні для класичних автоматизацій.

У цьому контексті основна досліджувана проблема полягає в тому, що на ринку майже відсутні доступні та відкриті системи, які не просто реагують на сигнали від сенсорів, а здатні осмислено адаптуватися до змін у поведінці

користувача, поточному контексті середовища чи зовнішніх умовах. Бракує рішень, які б поєднували переваги локального контролю з інтелектуальним аналізом — без складного навчання моделей або залежності від комерційної інфраструктури.

Метою цього дослідження є розробка прототипу адаптивної системи управління опаленням, яка поєднує можливості відкритої платформи Home Assistant із зовнішнім інтелектуальним модулем на базі LLM. Такий підхід дозволяє формувати рішення на основі реальних параметрів навколишнього середовища — температури, присутності, часу доби — та отримувати рекомендації у формі текстових повідомлень, що легко інтерпретуються або автоматично перетворюються на дії в межах системи розумного будинку.

Якість такої системи оцінюється з огляду на її здатність стабільно підтримувати температурні умови в межах комфортного діапазону, своєчасно реагувати на зміну ситуації в приміщенні, зменшувати потребу в ручному втручанні, а також зберігати можливість масштабування та налаштування без втрати контрольованості. Важливо також, щоб система залишалась автономною, працювала локально або з відкритим API, і не вимагала підключення до комерційних сервісів чи платформи.

Для реалізації поставленої мети необхідно було визначити відповідну архітектуру системи, підібрати інструменти для обміну даними між її частинами, забезпечити механізм взаємодії з LLM-модулем, розробити код, який буде відповідати за обробку запитів і відповідей, а також перевірити, як така система поводиться у різних змодельованих ситуаціях. Оскільки модель навчати не передбачалося, важливим аспектом стало правильне формулювання запитів та інтерпретація відповідей, а також обробка результатів у вигляді, придатному для подальшої автоматизації в Home Assistant.

Таким чином, дослідження орієнтоване на створення адаптивної та відкритої системи, яка поєднує зручність сучасних мовних моделей з гнучкістю відкритого програмного забезпечення й може бути впроваджена у різноманітних сценаріях Smart Home без залежності від сторонніх сервісів.

1.5 Обґрунтування доцільності створення власної системи

Після детального аналізу існуючих рішень у сфері автоматизованого управління мікрокліматом, як комерційних, так і відкритих, стало очевидно, що більшість із них мають суттєві функціональні або концептуальні обмеження. Комерційні продукти, зокрема Google Nest, Tado, Netatmo, Honeywell та інші, активно просуваються як готові до використання рішення, які забезпечують базовий комфорт та енергоефективність за рахунок встановлених сценаріїв та мобільних застосунків. Вони мають добре опрацьовані інтерфейси, високу стабільність роботи в типовому середовищі та швидку інтеграцію з найпоширенішими платформами.

Однак попри ці переваги, такі системи найчастіше побудовані як закриті екосистеми з обмеженим доступом до глибинних налаштувань, і критично залежать від постійного підключення до хмарних серверів виробника. У багатьох випадках вони не дозволяють повноцінного локального керування, не підтримують роботу без Інтернету або мають обмежений функціонал без платної підписки. Крім того, можливості інтеграції з нестандартними сенсорами або сторонніми автоматизаціями є вкрай обмеженими або взагалі відсутні. Користувач таких систем фактично опиняється в ситуації, коли доступ до його ж даних, журналів або керування обігрівом визначається зовнішніми умовами — сервісною політикою, регіональними обмеженнями або навіть поточним тарифом. Це суперечить концепції автономного і повністю контрольованого розумного простору, де саме користувач має остаточне слово у всьому, що відбувається в його домі.

На тлі цих обмежень відкриті платформи, як-от Home Assistant, OpenHAB або Domoticz, демонструють кардинально інший підхід. Вони орієнтовані не на споживання готового продукту, а на створення власного рішення. Home Assistant, зокрема, забезпечує найширші можливості для кастомізації — від побудови інтерфейсів і сценаріїв до тонкого налаштування логіки автоматизацій, інтеграцій із тисячами пристроїв і публічного API. Завдяки активній спільноті та

постійному розвитку екосистеми, ця платформа дозволяє побудувати систему, яка буде не лише зручною, а й відкритою, гнучкою, масштабованою та локалізованою під конкретне середовище.

Разом із тим, важливо усвідомлювати, що відкриті платформи вимагають від розробника певного рівня технічної підготовки. Користувачеві доводиться самостійно реалізовувати інтеграції, писати YAML-конфігурації, тестувати сценарії та перевіряти їх у реальному часі. Однак саме в цьому й полягає ключова перевага: замість обмеженого функціоналу — абсолютна свобода реалізації, з можливістю враховувати навіть нестандартні параметри, як-от рівень CO_2, час доби, календарні події або поведінкові шаблони.

Справжнім проривом у цьому контексті стало стрімке поширення та еволюція великих мовних моделей (LLM), які дозволяють аналізувати опис ситуації мовою, близькою до людської, і формувати контекстно релевантні відповіді. Замість класичного підходу типу «якщо температура нижче 20°C — увімкни опалення», можна передати опис умов у вигляді текстового запиту, а модель самостійно сформує обґрунтовану рекомендацію, враховуючи всі вхідні параметри. Такі моделі не потребують локального навчання — вони вже навчені на великому обсязі даних, і користувач працює з ними через API. З’являється можливість перенести логіку аналізу за межі локального обчислення, а натомість зосередитись на інтеграції — як саме отримати релевантну пораду та як перетворити її на дію в системі автоматизації.

У випадку розробки системи на базі Home Assistant із зовнішнім інтелектуальним модулем, як-от Google Gemini, реалізується не просто кастомізоване рішення — створюється прототип адаптивної системи нового покоління. Вона здатна реагувати не лише на фактичні значення, а й на зміни динаміки, на контекст ситуації, і навіть на поведінкові особливості користувача. Завдяки такому підходу можна поступово створювати справжній "цифровий термостат", який не просто вмикає або вимикає опалення, а обирає момент, режим і інтенсивність на основі узагальненої ситуаційної оцінки.

У підсумку, розробка власної системи з відкритими технологіями та

зовнішнім інтелектуальним модулем є не лише логічно обґрунтованою, але й практично доцільною. Вона поєднує високий ступінь автономії, можливість гнучкого налаштування, мінімальну залежність від сторонніх сервісів і потенціал для подальшого розвитку в напрямі повноцінного інтелектуального керування мікрокліматом. Саме тому такий підхід відкриває нові горизонти для побудови сучасних Smart Home-систем, які є не просто автоматизованими, а по-справжньому адаптивними, контекстно чутливими й орієнтованими на користувача.

1.6 Висновки

У першому розділі було здійснено всебічний аналіз поточного стану розвитку систем адаптивного керування опаленням. Розглянуто переваги та недоліки як комерційних рішень, що пропонуються провідними виробниками, так і відкритих платформ, таких як Home Assistant, OpenHAB і Domoticz. Особливу увагу приділено порівнянню традиційних підходів до автоматизації, заснованих на фіксованих правилах, із сучасними інтелектуальними методами, які передбачають врахування контексту, поведінки користувача та змін у навколишньому середовищі.

Було встановлено, що відкриті програмні платформи, за умови належного налаштування, мають значно більший потенціал для створення гнучких, кастомізованих і локально контрольованих систем. У той же час, їх використання потребує глибшого розуміння архітектури системи, принципів побудови логіки автоматизацій і засобів інтеграції зовнішніх інтелектуальних модулів. Саме тому вибір Home Assistant як основної платформи для реалізації проєкту є обґрунтованим, оскільки він поєднує гнучкість, масштабованість і активну спільноту підтримки.

У межах розділу також було обґрунтовано доцільність створення власної адаптивної системи на базі поєднання відкритого середовища та зовнішнього інтелектуального модуля, реалізованого на основі великої мовної моделі (LLM). Такий підхід дозволяє не лише реагувати на конкретні події, а й здійснювати

семантичний аналіз ситуацій та приймати рішення на основі природномовних описів умов.

На основі проведеного аналізу сформульовано мету дослідження, визначено критерії ефективності та окреслено конкретні задачі, реалізація яких дозволить побудувати прототип адаптивної системи управління опаленням нового покоління. Це створює логічне підґрунтя для переходу до практичної частини роботи — архітектурного проєктування, програмної реалізації та моделювання роботи системи.

2 РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ АДАПТИВНОГО ОПАЛЕННЯ

2.1 Архітектура системи: компоненти, принципи взаємодії, середовище виконання

У межах розробки адаптивної системи керування опаленням було прийнято рішення побудувати архітектуру на основі відкритої платформи Home Assistant. Цей вибір продиктований необхідністю досягнення високого рівня гнучкості, розширюваності та сумісності з сучасними інструментами інтелектуального аналізу даних. Архітектура системи проєктувалась із урахуванням трьох ключових принципів: локальна автономність, підтримка інтеграції зі сторонніми сервісами та можливість адаптації до змін у середовищі та поведінці користувачів.

У структурному плані система складається з п’яти основних функціональних блоків, кожен із яких виконує специфічну роль у загальному процесі. Першим компонентом виступають сенсори — пристрої, які відповідають за збір первинної інформації з фізичного середовища. До них відносяться температурні датчики, сенсори присутності, рівня CO_2, вологості, а також віртуальні джерела даних, як-от прогноз погоди або календар. Дані з цих сенсорів надходять у центральний вузол у вигляді структурованих повідомлень, як правило — у форматі JSON через протокол MQTT.

Наступним компонентом є виконавчі пристрої або актуатори. До них належать, наприклад, розумні термоголовки, електричні реле, контролери котлів або інші пристрої, здатні безпосередньо впливати на стан опалювальної системи. Вони отримують команди на основі оброблених даних або рекомендацій, які надходять із центрального модуля керування.

Ключовим елементом архітектури є Home Assistant — універсальний центр обробки подій, логіки автоматизації та візуалізації даних. Платформа розгортається локально, наприклад, на Raspberry Pi або повноцінному сервері, та відповідає за координацію взаємодії між усіма іншими модулями. Саме тут

реалізовано логіку автоматизацій, визначено правила тригерів та сценарії реагування. Home Assistant також забезпечує інтерфейс взаємодії з користувачем — через вебінтерфейс, мобільний застосунок або голосові асистенти.

Окремим та інноваційним елементом є інтеграція інтелектуального модуля, який реалізований як зовнішній скрипт на Python. Цей модуль приймає вхідні дані (температура, присутність, рівень CO_2, час доби) і формує текстовий запит до великої мовної моделі Gemini через API. У відповідь отримується рекомендація, яка інтерпретується як команда або пояснення, і повертається назад у систему у вигляді MQTT-повідомлення. Такий підхід дозволяє винести логіку прийняття рішень за межі базової платформи і делегувати її сучасній LLM, що аналізує контекст у природній мові.

Важливим компонентом системи є також інтерфейс користувача. Він дає змогу не лише переглядати поточні дані, але й вручну ініціювати сценарії, тестувати логіку, перевіряти відповіді інтелектуального модуля та спостерігати за історією подій. Користувач має повний контроль над усіма параметрами, а за необхідності — може адаптувати систему під свої потреби без залучення сторонніх сервісів.

Взаємодія між усіма компонентами побудована на основі стандартних протоколів. Основний — MQTT, який забезпечує двосторонній обмін повідомленнями між Home Assistant і зовнішнім скриптом. Для інтеграції з мовною моделлю використовується HTTPS-запит до API сервісу Gemini. Інші частини платформи можуть використовувати REST API, WebSocket або локальні інтеграції. Це дозволяє легко масштабувати архітектуру, додаючи нові функції або джерела даних без необхідності перебудови системи в цілому.

Виконання системи повністю локалізоване. Усі компоненти, окрім звернення до зовнішньої LLM, функціонують без підключення до Інтернету, що гарантує стабільність, швидкодію та конфіденційність. У разі відсутності зв’язку із зовнішнім API, система може працювати у fallback-режимі або переходити до простих локальних автоматизацій.

Загальну логічну структуру системи подано на діаграмі компонентів UML

(рис 2.1), яка ілюструє основні модулі системи, їхню взаємодію між собою, а також зовнішні сервіси, такі як MQTT-брокер та інтелектуальний модуль на базі Google Gemini.

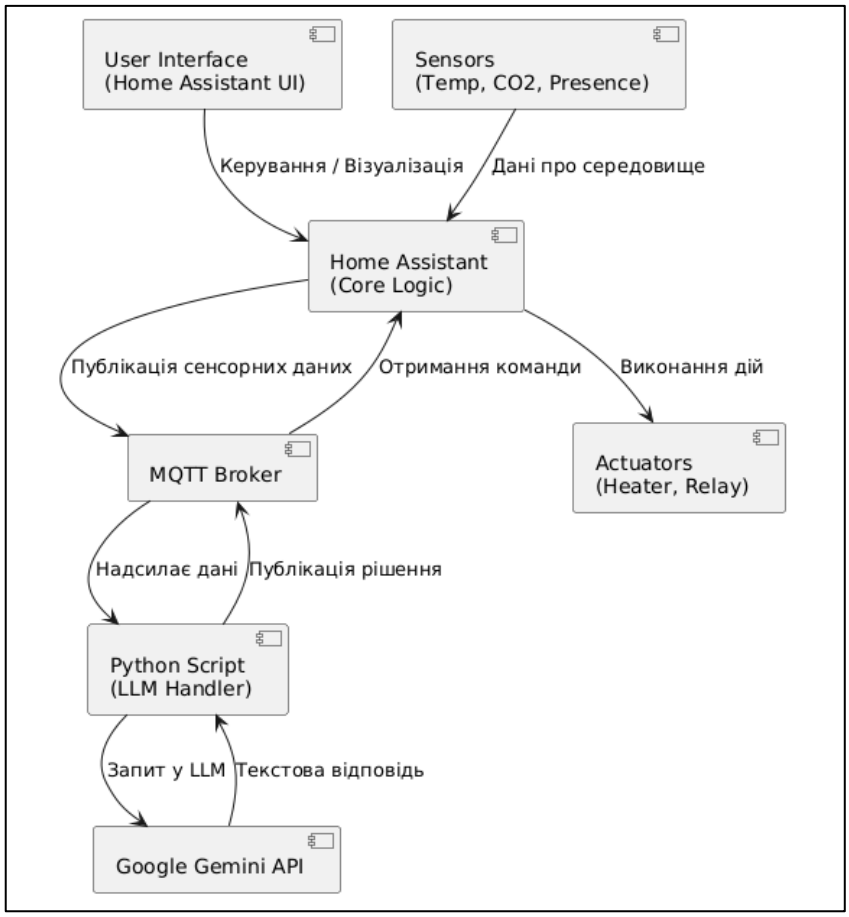


Рисунок 2.1 – UML-діаграма компонентної архітектури адаптивної системи опалення на базі Home Assistant

2.2 Моделювання архітектури адаптивної системи засобами UML

Для формального опису логіки функціонування адаптивної системи опалення було використано уніфіковану мову моделювання UML. Такий підхід дає змогу візуально представити компоненти системи, їхню взаємодію, алгоритмічну послідовність дій, а також сценарії взаємодії з користувачем і зовнішніми сервісами. Це особливо важливо у випадках, коли система включає декілька незалежних рівнів — сенсорний, логічний, виконавчий і зовнішній інтелектуальний модуль, з яким комунікація здійснюється через API.

З метою комплексного представлення архітектури системи було створено чотири типи UML-діаграм: варіантів використання, класів, послідовностей і діяльності. Кожна з них відображає окрему грань функціонування системи — від зовнішньої взаємодії до внутрішньої логіки обробки запитів.

Першою було побудовано діаграму варіантів використання, що відображає основні ролі та сценарії взаємодії користувача з системою. На рисунку 2.2 показано, що мешканець має змогу ініціювати автоматизації, переглядати дані сенсорів, отримувати зворотний зв’язок у вигляді текстових рекомендацій, а також вручну запускати окремі сценарії. Система Home Assistant реагує як на дії користувача, так і на вхідні сигнали від сенсорів, публікуючи дані у MQTT та очікуючи відповідь від зовнішнього Python-скрипта, який, у свою чергу, звертається до мовної моделі LLM (наприклад, Gemini).

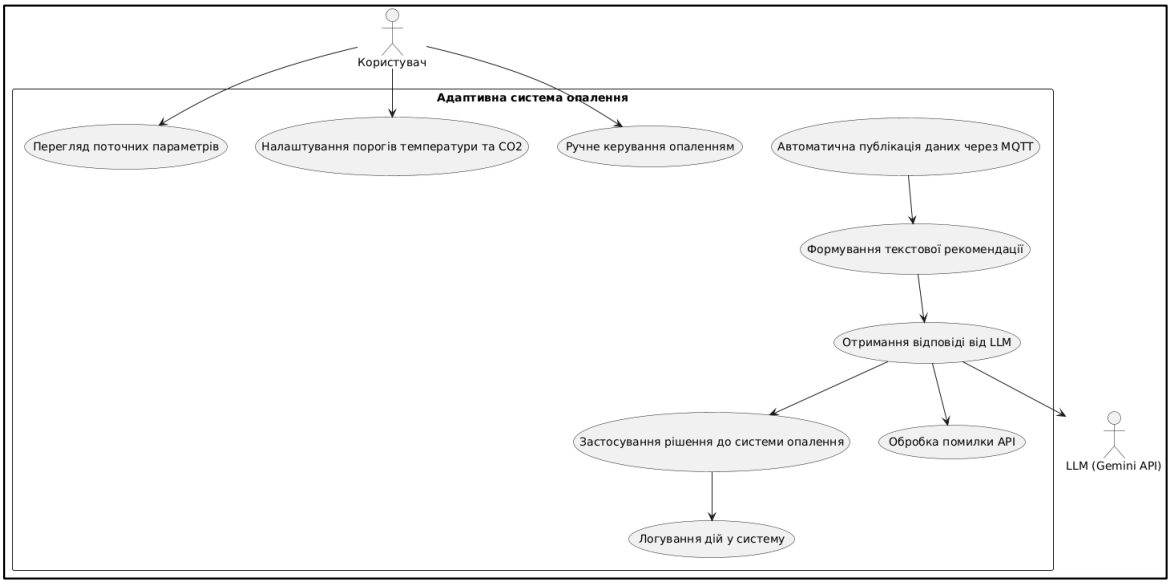


Рисунок 2.2 – Варіанти використання з урахуванням взаємодії з мовною моделлю

Наступним етапом моделювання стала побудова діаграми класів. На рисунку 2.3 відображено основні логічні компоненти системи — об’єкти, які відповідають за обробку сенсорних даних, публікацію та підписку через MQTT, формування запитів до LLM, зберігання кешованих відповідей, логування прийнятих рішень та відображення результатів у графічному інтерфейсі. Хоча

програмна реалізація системи побудована на основі процедурного підходу, діаграма класів допомагає структуровано представити функціональні блоки системи та їхню взаємодію у вигляді логічних сутностей.

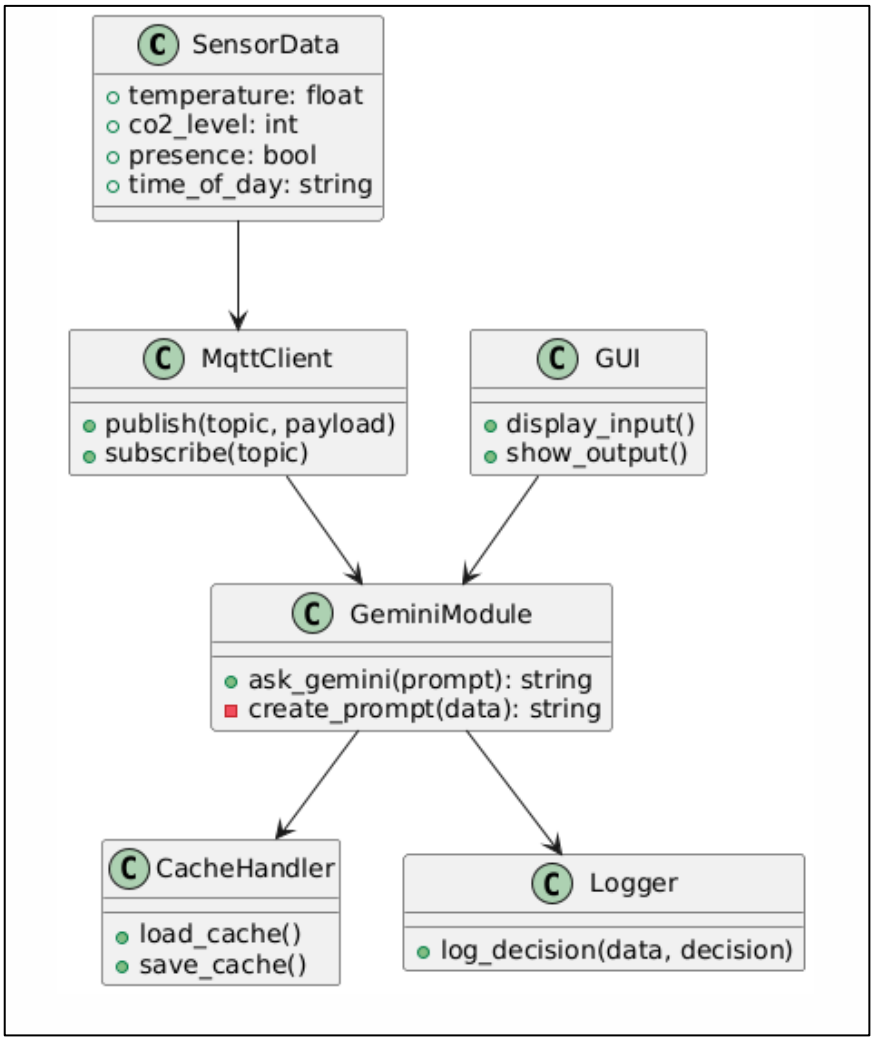


Рисунок 2.3 – UML-діаграма класів адаптивної системи

Для відображення динаміки обміну повідомленнями в системі побудовано діаграму послідовностей. На рисунку 2.4 змодельовано типовий сценарій, у якому Home Assistant надсилає сенсорні дані через MQTT-брокер, Python-скрипт отримує ці дані, формує запит до LLM, отримує текстову рекомендацію, після чого публікує відповідь назад у систему, де вона інтерпретується як команда для виконавчих пристроїв. Така діаграма чітко демонструє взаємозв’язок між усіма компонентами в реальному часі.

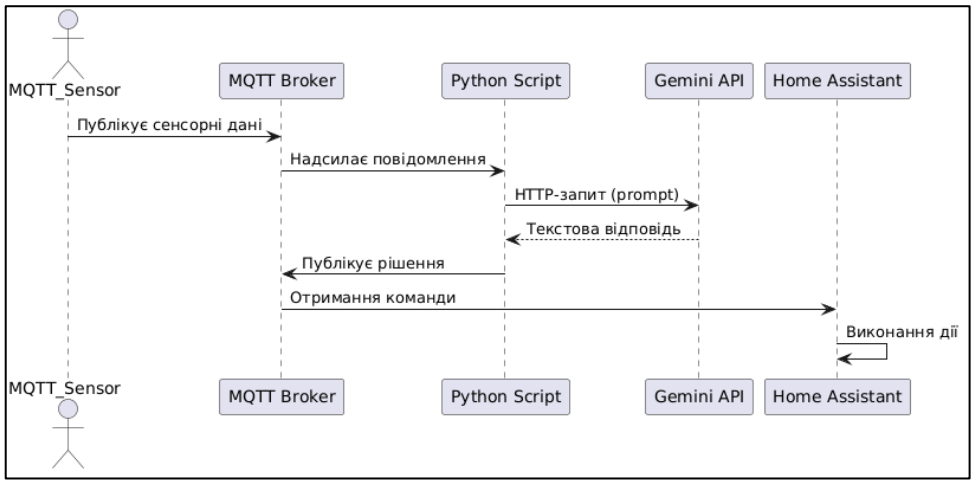


Рисунок 2.4 – UML-діаграма послідовностей обробки сенсорних даних

Завершальним елементом моделювання стала побудова діаграми діяльності, яка ілюструє логіку прийняття рішень Python-скриптом на основі даних з MQTT. На рисунку 2.5 показано покроковий алгоритм: перевірка наявності відповіді в кеші, формування запиту у разі її відсутності, взаємодія з LLM, обробка відповіді та публікація результату. Додатково враховано ситуації обробки винятків, зокрема випадки, коли зовнішній сервіс недоступний. Така схема дозволяє виявити потенційні вузькі місця в логіці та забезпечити стабільність роботи системи за рахунок резервних шляхів виконання.

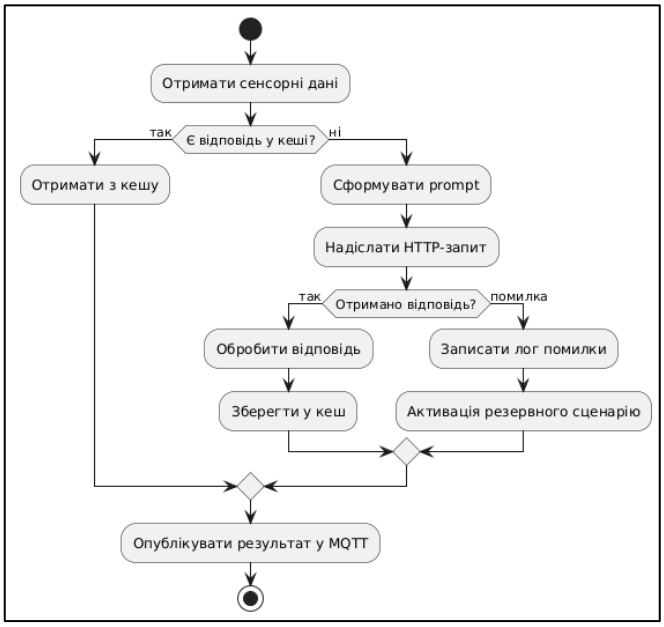


Рисунок 2.5 – UML-діаграма діяльності для обробки запиту до мовної моделі

У підсумку, застосування UML-діаграм дозволило структуровано відобразити як архітектуру адаптивної системи опалення, так і сценарії її роботи в динаміці. Кожна з діаграм виконує свою роль у поясненні функціонування системи, що суттєво полегшує її розуміння, обслуговування, тестування та розвиток. Усі ключові механізми — від збору даних до формування рішень за допомогою зовнішньої мовної моделі — були формалізовані у вигляді наочних графічних моделей, що є невід'ємною частиною технічної документації системи.

2.3 Вибір оптимальної IT-інфраструктури

Для ефективного функціонування адаптивної системи опалення було сформовано гнучку інформаційно-технічну інфраструктуру, здатну працювати як у локальному режимі, так і з використанням зовнішніх інтелектуальних сервісів. Основна вимога до архітектури — забезпечити стабільний обмін даними між сенсорами, модулем обробки рішень та Home Assistant без залежності від хмарного середовища. Це дозволяє досягти автономності системи при збереженні можливості підключення до потужних зовнішніх мовних моделей.

Центральним елементом системи є одноплатний комп'ютер Raspberry Pi 4, який виконує роль локального сервера. Саме на ньому встановлено Home Assistant, MQTT-брокер, а також Python-скрипт, що обробляє сенсорні дані, формує запит у вигляді текстового опису ситуації та надсилає його до великої мовної моделі Gemini через офіційний API. Отримана відповідь інтерпретується як рекомендація, яка знову передається через MQTT назад у Home Assistant для подальших дій.

Уся інфраструктура умовно складається з кількох взаємозалежних блоків. Сенсорний рівень включає фізичні пристрої, які вимірюють температуру повітря, рівень CO_2, присутність у приміщенні, а також додаткові показники, які можуть бути змодельовані або зчитані з API. Комунікація між сенсорами та сервером відбувається через протоколи Zigbee або MQTT, залежно від

конкретної реалізації пристроїв. Зібрані дані надходять до Home Assistant і паралельно публікуються у MQTT-топіки для обробки Python-скриптом.

Виконавчий рівень представлений актуаторами — розумними реле, термостатичними клапанами, модулями керування котлом або електронагрівачем. Команди до цих пристроїв надходять із Home Assistant, який або самостійно визначає логіку дій, або орієнтується на рекомендації, отримані від зовнішньої мовної моделі.

Аналітичний рівень реалізовано як окремий модуль на Python. Він виступає посередником між сенсорними даними та LLM. При отриманні нових даних модуль перевіряє наявність відповідної відповіді в кеші, і лише за її відсутності надсилає запит до Gemini. Таким чином, зменшується навантаження на зовнішній API та підвищується швидкодія системи. Вся логіка — від формування запиту до публікації рішення — реалізована у вигляді єдиного, інтегрованого скрипта.

Особливу увагу приділено локальності інфраструктури: базові функції продовжують працювати навіть без доступу до інтернету. MQTT, Home Assistant, графічний інтерфейс, історія подій — усе функціонує автономно. Мовна модель, у свою чергу, виступає як зовнішнє розширення для розумнішого аналізу ситуації, але не є критичною для базової життєдіяльності системи.

Для взаємодії з користувачем передбачено два режими: веб-інтерфейс Home Assistant (на основі Lovelace UI) та локальний графічний інтерфейс на основі tkinter. Перший дозволяє бачити загальний стан системи, а другий — вручну вводити тестові параметри та бачити, яку відповідь сформує мовна модель. Усі рішення, що генеруються LLM, автоматично логуються у CSV-файл, а історія значень сенсорів зберігається в базі даних Home Assistant або в зовнішніх аналітичних системах, таких як InfluxDB.

Архітектура проєкту залишається модульною і масштабованою. Додавання нових типів сенсорів або сценаріїв автоматизації не потребує переписування всієї логіки, а реалізується через оновлення YAML-конфігурацій або розширення Python-обробки. Завдяки цьому система легко адаптується під

нові задачі, наприклад, контроль за вологістю, керування вентиляцією або розширення на інші приміщення. Структуру взаємодії основних компонентів системи наведено на рисунку 2.6.

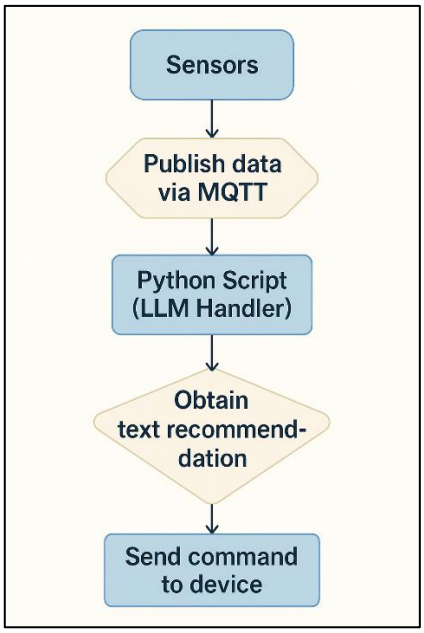


Рисунок 2.6 – Схема логіки прийняття рішень у системі на основі сенсорних даних і LLM

2.4 Вибір апаратного та програмного забезпечення, інтеграція з Home Assistant

Після побудови загальної архітектури системи наступним кроком став підбір апаратних та програмних компонентів, які мали б забезпечити стабільне, автономне й адаптивне функціонування системи управління опаленням. Основна увага при цьому приділялась таким критеріям, як: сумісність з Home Assistant, підтримка протоколів автоматизації (MQTT, Zigbee), простота інтеграції з зовнішніми інтелектуальними сервісами, стабільність роботи в локальній мережі без постійної залежності від хмари.

У ролі центрального обчислювального вузла було обрано мікрокомп’ютер Raspberry Pi 4. Це рішення забезпечує оптимальний баланс між продуктивністю, енергоефективністю та вартістю. На Raspberry Pi розгорнуто операційну систему

Home Assistant OS, яка надає повноцінне середовище для керування автоматизацією, обробки вхідних сигналів, запуску додаткових сервісів та взаємодії з іншими пристроями. Крім того, на цьому ж пристрої функціонує окремий Python-скрипт, який реалізує логіку взаємодії з великою мовною моделлю (LLM) через зовнішній API від Google Gemini.

Сенсори, які входять до складу системи, збирають дані про навколишнє середовище — температуру повітря, рівень CO_2, наявність або відсутність людей у приміщенні. Ці пристрої можуть бути як Zigbee-сумісними (наприклад, температурні датчики Xiaomi або Sonoff), так і працювати безпосередньо через MQTT. З'єднання Zigbee забезпечується через USB-стик (наприклад, Sonoff ZBDongle-E), який підключається до Raspberry Pi і дозволяє інтегрувати декілька бездротових пристроїв одночасно.

Для впливу на мікроклімат застосовуються різні актуатори: це можуть бути розумні реле (Sonoff, Tuya), які вмикають або вимикають нагрівальні пристрої, або термостатичні головки на батареях, які регулюють тепловіддачу. Усі ці пристрої легко інтегруються з Home Assistant і можуть бути пов'язані з автоматизаціями, які реагують на зміну стану середовища або команди, отримані від LLM.

Комунікація між сенсорами, центральною системою та Python-скриптом реалізована через брокер MQTT, який встановлено локально на Raspberry Pi. Коли сенсори надсилають нові показники, Home Assistant публікує їх у відповідному топіку. Python-скрипт, який працює незалежно від Home Assistant, підписується на цей топік, отримує дані, формує текстовий запит (prompt) та надсилає його до зовнішньої мовної моделі Google Gemini. Відповідь у вигляді текстової рекомендації публікується назад у зворотний топік, після чого Home Assistant використовує цю інформацію для прийняття рішень — наприклад, вмикання або вимикання опалення.

На додаток до вебінтерфейсу Home Assistant, система також включає простий графічний інтерфейс, реалізований через бібліотеку tkinter. Цей інтерфейс призначений для ручного введення параметрів, тестування взаємодії з

LLM або виведення відповідей, що дозволяє швидко перевірити, як система реагує на зміну умов без необхідності використовувати реальні сенсори.

Оскільки система не використовує локального навчання моделей машинного навчання, немає необхідності у встановленні спеціалізованих бібліотек типу TensorFlow чи PyTorch. Вся логіка інтелектуальної частини передається на зовнішній сервіс, а запит формується в режимі реального часу та надсилається за допомогою бібліотеки requests. Відповідь обробляється локально та, за необхідності, кешується, щоб уникнути дублювання запитів.

Інтеграція між Home Assistant та Python-модулем є повністю модульною. Якщо виникне потреба замінити сервіс (наприклад, перейти з Gemini на інший LLM), це можна зробити без змін основної логіки Home Assistant. Така структура гарантує масштабованість: у майбутньому можна буде додавати інші сервіси — наприклад, прогноз погоди або локальні правила на основі графіка користувача.

Всі обчислювальні та автоматизовані дії логуються — зокрема, рішення від LLM записуються у CSV-файл, а Home Assistant автоматично веде історію подій, яку можна переглядати в інтерфейсі або через сторонні сервіси (наприклад, Grafana або InfluxDB). Візуалізацію загальної взаємодії між основними компонентами системи наведено на рисунку 2.7.

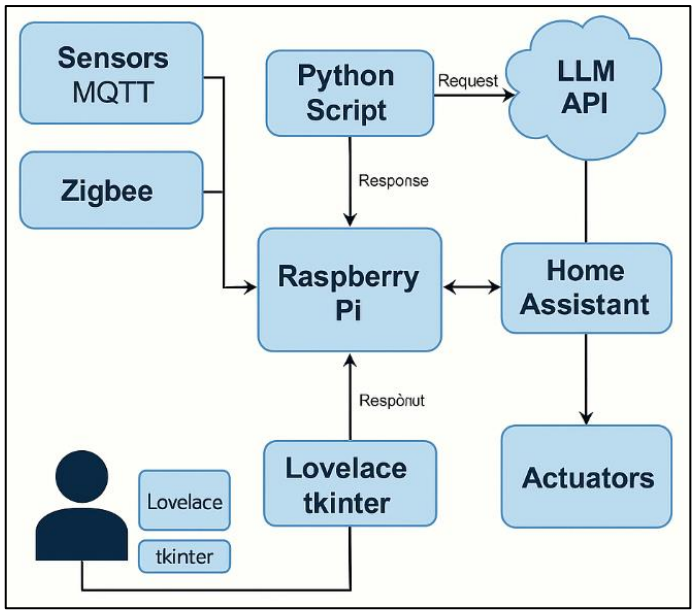


Рисунок 2.7 – Компонентна схема апаратного та програмного забезпечення системи

Таким чином, вибрані апаратні й програмні компоненти забезпечують не лише реалізацію базової автоматизації, але й дозволяють створити адаптивну, розширювану, стабільну та відкриту платформу для подальшого розвитку системи опалення в межах концепції “розумного будинку”.

2.5 Реалізація автоматизацій, логіка прийняття рішень, взаємодія з інтелектуальним модулем

У розробленій системі керування опаленням центральним елементом прийняття рішень виступає інтелектуальний модуль, реалізований у вигляді Python-скрипта з інтеграцією великої мовної моделі Gemini від Google. На відміну від класичних підходів, де сценарії роботи базуються на чітко визначених умовах або шаблонах, запропонована реалізація дозволяє враховувати контекст і формувати рекомендації на основі вхідних параметрів у природній мові.

Збір вихідних даних здійснюється за допомогою ручної публікації у MQTT-топік home/sensors через вбудований інтерфейс Home Assistant. У якості джерел використовуються змодельовані значення температури, рівня CO_2, факту присутності та часу доби. Таким чином, система не залежить від реального обладнання й дозволяє відлагоджувати логіку на етапі теоретичної розробки.

Отримавши повідомлення, інтелектуальний модуль формує текстовий запит, що описує поточний стан середовища. Наприклад:

«Поточна температура: 21.5°C. Рівень CO_2: 950 ppm. Присутність у кімнаті: так. Час доби: 18:30. Які дії слід виконати в системі опалення?»

Цей запит надсилається до мовної моделі через API-запит. У відповідь модель генерує текстову рекомендацію, яка потім публікується в MQTT-топік home/heating/command. Вихідна інформація містить не формалізовану команду, а повноцінне текстове пояснення, яке за потреби можна обробити або використати в подальших автоматизаціях.

Home Assistant у цій версії не виконує обробку цієї відповіді автоматично.

Наразі система працює в демонстраційному режимі, де симуляція сценаріїв та прийняття рішень відбувається через тестові MQTT-повідомлення, а інтеграція з діями Home Assistant залишається потенційною — її можна реалізувати шляхом додавання тригерів на основі ключових слів або шаблонів.

Щоб уникнути дублювання звернень до API при повторенні однакових параметрів середовища, у скрипті реалізовано кешування. Усі сформовані запити та відповідні відповіді зберігаються у локальному JSON-файлі, що дозволяє зменшити кількість викликів до мовної моделі та прискорити роботу.

Окрім автоматичного режиму взаємодії через MQTT, у скрипті реалізовано альтернативний варіант — запуск у графічному режимі. Через графічний інтерфейс, створений засобами бібліотеки Tkinter, користувач може вручну ввести значення параметрів середовища, сформувати запит і миттєво отримати відповідь. Це особливо зручно для демонстрацій або тестування логіки без підключення до брокера MQTT.

Усі відповіді мовної моделі разом із відповідними параметрами зберігаються в лог-файл у форматі CSV. Це дозволяє згодом виконати базову статистичну оцінку — наприклад, частоту рекомендацій або середні значення вхідних параметрів на момент формування рішень.

Підсумовуючи, запропонована реалізація забезпечує гнучку архітектуру, де логіка прийняття рішень виноситься за межі системи автоматизації та делегується зовнішньому інтелектуальному модулю. Незважаючи на те, що система наразі працює в умовах теоретичного моделювання, її структура повністю готова до розширення й практичного впровадження у розумне середовище з реальними сенсорами та виконавчими пристроями.

2.6 Приклади реалізованих сценаріїв автоматизації та візуалізація в Home Assistant

Оскільки в межах даної роботи система розроблялася на етапі теоретичної апробації, основна увага була зосереджена на моделюванні сценаріїв роботи, а

не на повноцінному впровадженні у фізичне середовище з сенсорами та автоматизованими діями. Усі приклади реалізовані шляхом ручної взаємодії з MQTT-брокером через інтерфейс розробника Home Assistant.

Для тестування логіки було змодельовано кілька сценаріїв, які імітують зміни параметрів у середовищі — зокрема, комбінації температури, рівня CO_2 та присутності людей. Дані надсилалися у вигляді JSON-повідомлень у топік home/sensors, який прослуховує Python-скрипт із вбудованим модулем звернення до мовної моделі Gemini.

Кожен змодельований сценарій охоплював наступні дії: спочатку повідомлення вручну надсилалося через Home Assistant, далі його обробляв зовнішній скрипт, після чого формувалась текстова рекомендація, яка публікувалася у зворотний MQTT-топік home/heating/command.

Приклад одного з таких сценаріїв:

```
json
CopyEdit
{
  "temperature": 19.0,
  "co2": 1200,
  "presence": true
}
```

У відповідь модель могла повернути: "Рекомендується тимчасово зменшити обігрів, оскільки рівень CO_2 є високим і свідчить про недостатнє провітрювання приміщення."

Ця відповідь зберігалася у журналі рішень і слугувала прикладом адаптивної реакції системи на змінені умови. Автоматичне застосування таких рекомендацій наразі не реалізовано, проте архітектура системи дозволяє легко інтегрувати цю функціональність через створення автоматизацій на основі вмісту MQTT-повідомлення.

У межах візуалізації Home Assistant використовувався переважно як інтерфейс для роботи з MQTT. Панелі, графіки або окремі інтерактивні карти не

створювалися, оскільки тестування обмежувалося логічною взаємодією між модулями. Незважаючи на це, структура платформи повністю дозволяє реалізувати візуальне представлення стану сенсорів, історії змін та рекомендацій у майбутніх ітераціях.

Таким чином, розроблена система на поточному етапі демонструє здатність аналізувати змодельовані сценарії, приймати текстові рішення на основі параметрів середовища та зберігати результати для подальшого аналізу. Це створює базу для розширення функціоналу — зокрема, впровадження автоматизованих дій у Home Assistant на основі рекомендацій, сформованих великою мовною моделлю.

2.7 Тестування системи, результати та аналіз ефективності

Оскільки розроблена система не передбачала повного розгортання в реальних умовах, тестування проводилося у симульованому середовищі з використанням штучно згенерованих даних. Усі сценарії перевірялися в теоретичному режимі — через ручну публікацію MQTT-повідомлень або моделювання подій за допомогою внутрішніх засобів Home Assistant.

Метою цього етапу було переконатися в тому, що основні компоненти системи — Home Assistant, MQTT-брокер, інтелектуальний модуль — взаємодіють між собою стабільно, а логіка прийняття рішень працює згідно з очікуваним сценарієм. Акцент робився не на фізичних реакціях системи (вмикання пристроїв), а на перевірці правильності обміну даними та обробки команд.

Під час перевірки були змодельовані найпоширеніші побутові ситуації, такі як температурні коливання в приміщенні, тривала відсутність присутності, підвищення концентрації CO_2, нічний час доби та фіксація руху при слабкому освітленні.

Усі події відтворювалися шляхом надсилання відповідного JSON-повідомлення в MQTT-топік home/sensors. У відповідь Python-скрипт формував текстову рекомендацію, яка надсилалася у топік home/heating/command. З боку

Home Assistant перевірялася лише здатність приймати повідомлення — повноцінна реакція системи (наприклад, увімкнення опалення) на основі цієї відповіді не налаштовувалася в рамках цієї роботи.

Особливу увагу було приділено стабільності взаємодії між модулями: повідомлення оброблялись без затримок і помилок, MQTT-зв’язок залишався стабільним, а мовна модель відповідала на запити у межах допустимого часу.

Ще одним аспектом стало тестування коректності генерації промптів і отриманих відповідей. Модель Gemini стабільно генерувала осмислені текстові рекомендації, релевантні до вхідних даних. Вони могли зберігатися в журналі або, у перспективі, використовуватися для подальших автоматизованих рішень у системі.

Результати моделювання свідчать, що навіть у симуляційному режимі система демонструє стійку роботу, а її логіка залишається прозорою, масштабованою та адаптованою до інтеграції з реальними пристроями в майбутньому.

2.8 Використані інструменти: практичні аспекти реалізації

У процесі реалізації програмної частини системи адаптивного опалення було використано низку інструментів, які разом формують єдиний технологічний ланцюг. Центральним середовищем для тестування і моделювання стало програмне забезпечення Home Assistant OS, яке було розгорнуто локально за допомогою віртуальної машини VirtualBox (рис. 2.8). Система була налаштована з виділенням 4 ГБ оперативної пам’яті та трьох віртуальних процесорних ядер, що забезпечувало її стабільну роботу у рамках дослідження.

Таке апаратне конфігурування виявилося достатнім для забезпечення стабільної, безперебійної роботи програмного середовища протягом усього дослідницького циклу. У процесі експериментів система демонструвала надійність і чітку реакцію на події в реальному часі, що свідчить про добре

збалансований розподіл ресурсів. Це середовище також дозволяло масштабувати систему за потреби, з можливістю збільшення обсягів пам’яті або кількості ядер у разі зростання навантаження чи переходу до складніших задач.

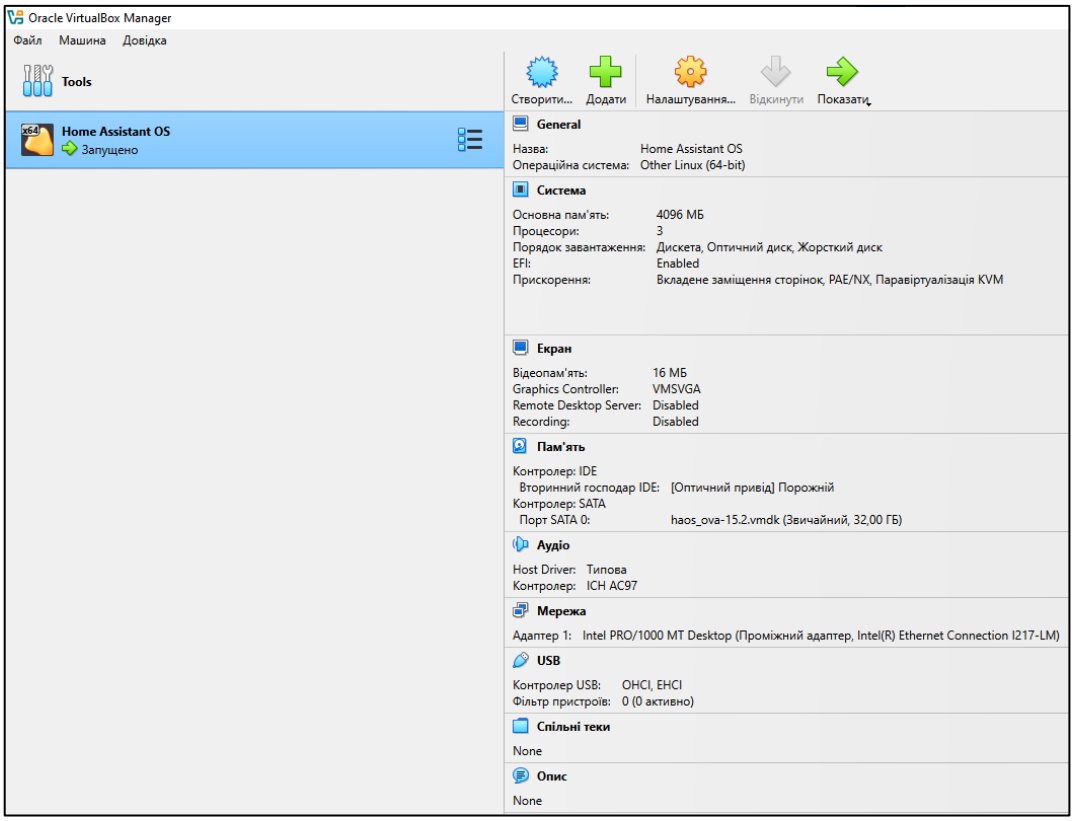


Рисунок 2.8 – Розгорнута віртуальна машина Home Assistant OS у середовищі VirtualBox

Доступ до функціоналу Home Assistant здійснювався через веб-інтерфейс, де проводилися симуляції сценаріїв взаємодії пристроїв. Зокрема, за допомогою вбудованих інструментів розробника було згенеровано тестові повідомлення у форматі JSON, які імітували дані від температурного сенсора, датчика CO_2 та сенсора присутності. Ці повідомлення вручну надсилались у топік home/sensors, що дозволило перевірити, як система реагує на зміну вхідних параметрів (рис. 2.9).

Таке тестування стало важливою частиною загального процесу перевірки функціональності системи, оскільки дало змогу не лише оцінити швидкість реагування на події, а й виявити потенційні недоліки в логіці автоматизації.

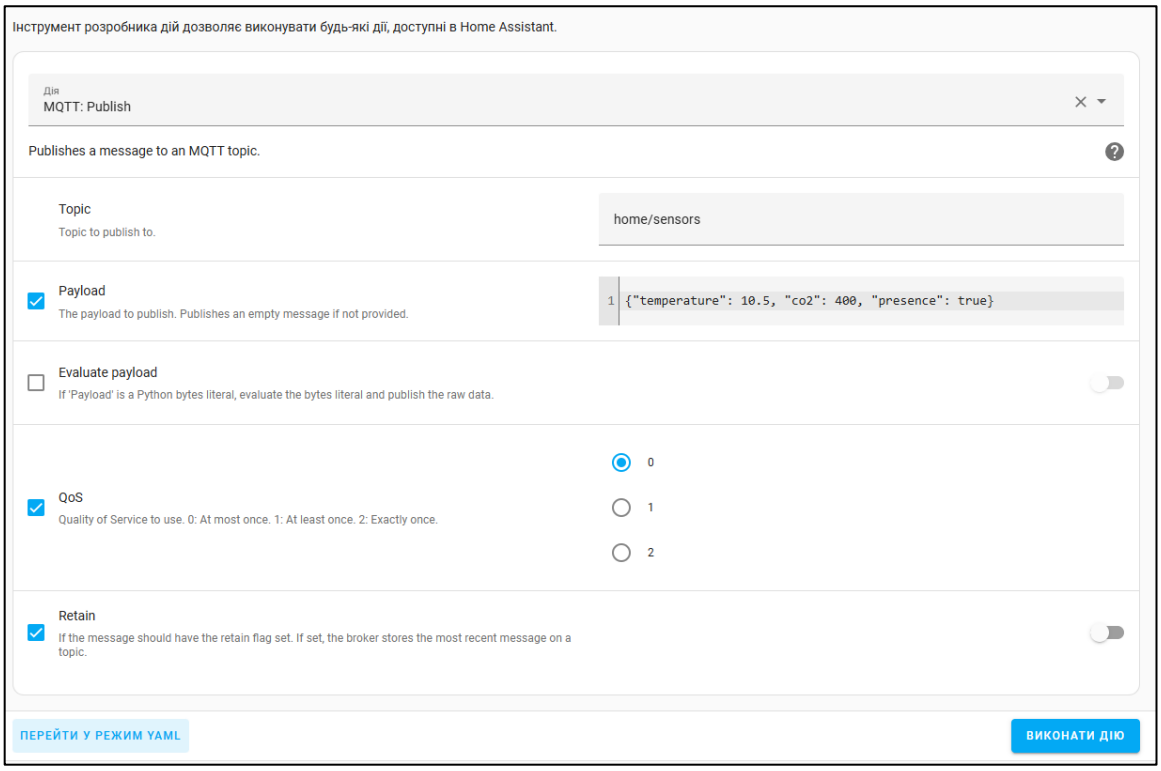


Рисунок 2.9 – Симуляція публікації MQTT-повідомлення з даними сенсорів у Home Assistant

Для забезпечення комунікації між Home Assistant та Python-скриптом використовувався протокол MQTT, реалізований через вбудовану інтеграцію Mosquitto broker. Цей брокер було активовано в самому Home Assistant, без використання сторонніх рішень. Підключення зі скрипта здійснювалось через внутрішню IP-адресу локальної мережі з використанням базової авторизації (рис. 2.10). Таким чином, забезпечувався повноцінний обмін даними між розумним будинком і інтелектуальним модулем.

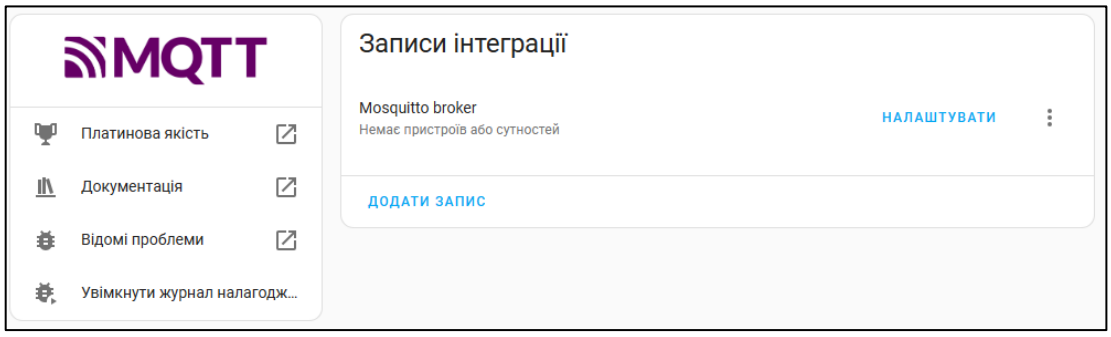


Рисунок 2.10 – Інтеграція MQTT-брокера Mosquitto у середовищі Home Assistant

Ключовим компонентом системи став модуль прийняття рішень, реалізований на базі Google Gemini API. Для цього було створено окремий проєкт у Google Cloud Console, у межах якого вручну активовано Generative Language API. Згенерований API-ключ використовувався у Python-скрипті для виконання запитів до моделі Gemini 1.5 Pro. Формування запитів відбувалося у форматі JSON з текстовим описом умов — температури, концентрації CO_2, присутності та часу доби. Модель повертала текстову рекомендацію, яка оброблялась у скрипті та публікувалась у зворотному топіку MQTT (рис. 2.11).

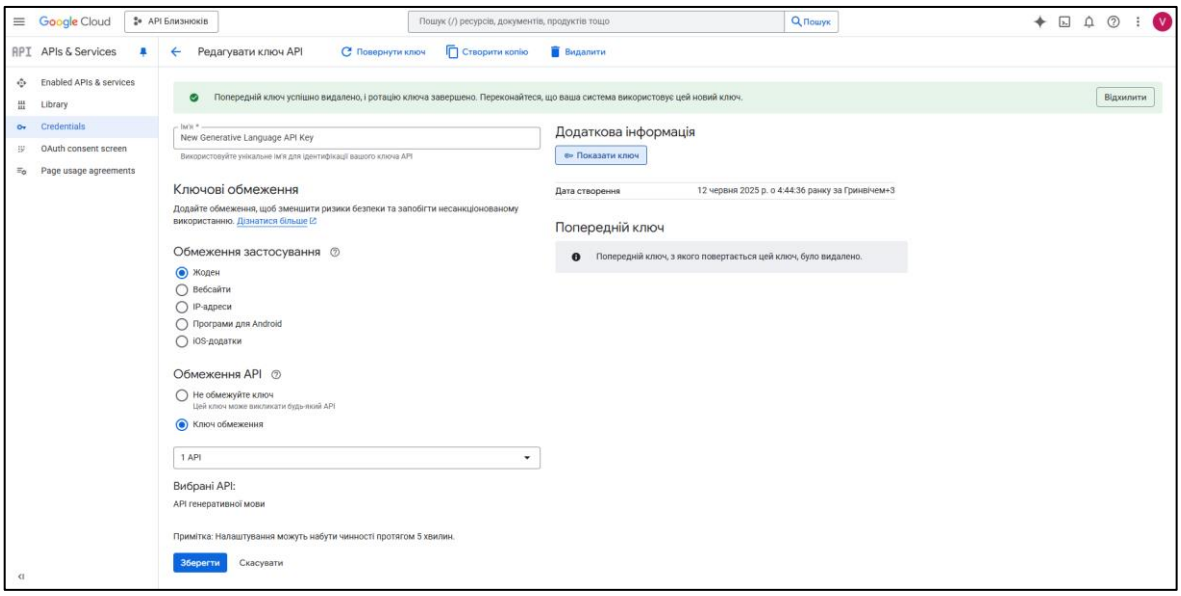


Рисунок 2.11 – Налаштування API-ключа Gemini

Уся логіка взаємодії та автоматизації була реалізована у скрипті intelligent_heating_control.py, що підтримує три режими роботи: mqtt — для автоматичної взаємодії з брокером, gui — для ручного введення даних через графічний інтерфейс tkinter, та stats — для перегляду накопиченої статистики рішень. Скрипт розроблявся у середовищі Visual Studio Code, запуск здійснювався через вбудований термінал (рис. 2.12). Розробка проводилася у глобальному Python-середовищі без використання venv, що не стало на заваді реалізації повного функціоналу.

Середовище залишалося стабільним, а всі необхідні бібліотеки успішно працювали без ізоляції.

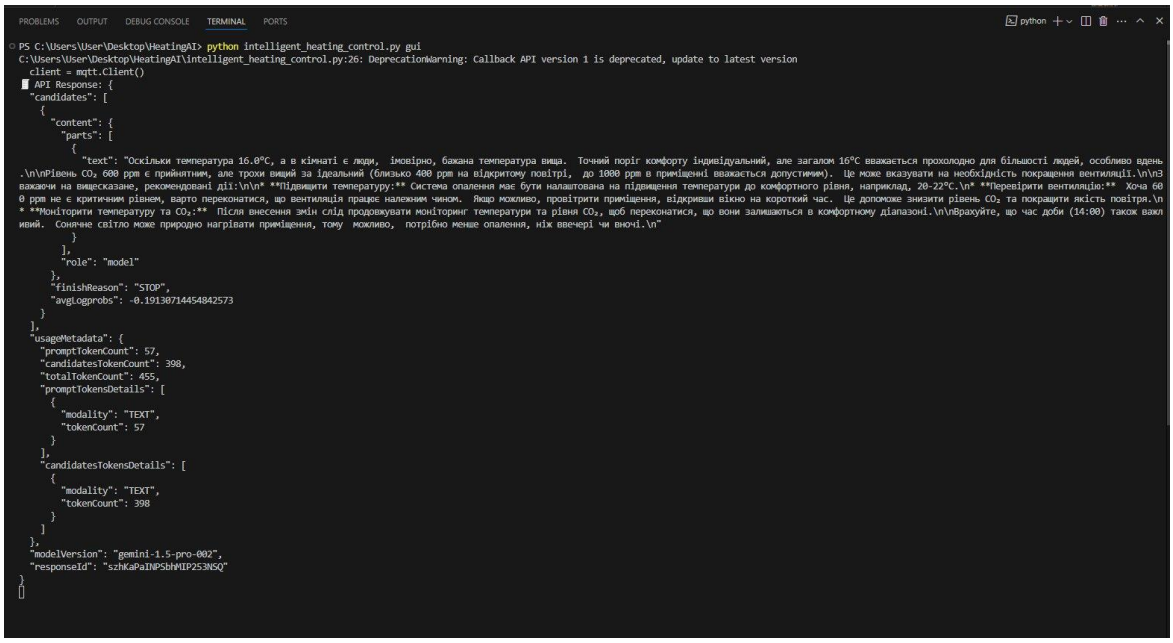


Рисунок 2.12 – Приклад запуску Python-скрипта системи керування опаленням у середовищі Visual Studio Code

Усі згенеровані рішення автоматично зберігались у файл ai_decisions.csv, що дозволяло здійснювати подальший аналіз через бібліотеку pandas. Крім того, для зменшення кількості звернень до API використовувався механізм кешування у форматі JSON (gemini_cache.json). Структура робочої директорії, представлена на рисунку 2.13, демонструє організацію файлів та логіку проєкту: тут знаходяться всі ключові компоненти — скрипти, журнали, кеш і лог-файли.

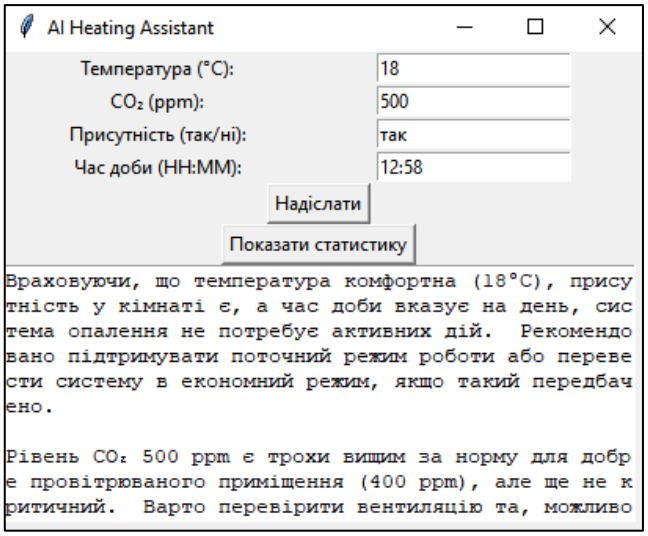


Рисунок 2.13 – Вивід рекомендацій ШІ у графічному інтерфейсі системи на основі введених параметрів

Таким чином, реалізація системи, хоч і в симуляційному середовищі, охоплює повний цикл роботи: від моделювання сенсорних даних до формування рішення на основі штучного інтелекту та подачі команди до середовища розумного будинку.

2.9 Висновки

У другому розділі було представлено проєкт адаптивної системи опалення з використанням платформи Home Assistant у поєднанні з інтелектуальним модулем, реалізованим у вигляді зовнішнього Python-скрипта. Визначено загальну архітектуру системи, обґрунтовано вибір програмного забезпечення та протоколу MQTT для взаємодії між компонентами. Особливу увагу приділено інтеграції з мовною моделлю Gemini, яка виконує функції аналізу ситуації та формування текстових рекомендацій на основі сенсорних даних.

У рамках моделювання були реалізовані приклади сценаріїв та протестована передбачувана логіка системи. Перевірено стабільність взаємодії між модулями, коректність обміну повідомленнями та формування релевантних відповідей мовною моделлю. Хоча повноцінна автоматизація дій у Home Assistant не впроваджувалась, архітектура системи дозволяє легко масштабувати її для роботи з реальними пристроями.

Система продемонструвала стабільну роботу в симуляційному режимі, гнучкість підходу до формування рішень, а також високий потенціал до подальшого вдосконалення — зокрема, шляхом автоматичного виконання AI-рекомендацій, розширення сенсорної бази та впровадження повноцінного користувацького інтерфейсу.

3 ТЕСТУВАННЯ, ОЦІНЮВАННЯ РЕЗУЛЬТАТІВ ТА ВИСНОВКИ

3.1 Методика тестування: сценарний підхід на основі моделювання

Для перевірки працездатності та оцінки адаптивної логіки системи інтелектуального керування опаленням було обрано сценарний підхід, заснований на моделюванні типових побутових ситуацій у віртуальному середовищі. Такий метод дозволив наблизити процес тестування до умов реального використання, не маючи фізичних сенсорів чи апаратного впровадження.

Тестування проводилося в кількох режимах роботи розробленого Python-скрипта, кожен з яких дозволяв оцінити окремі аспекти функціонування системи:

– MQTT-режим – взаємодія з Home Assistant через брокер MQTT. Скрипт автоматично приймає дані, згенеровані вручну або з інтерфейсу розробника, формує запит до мовної моделі та повертає рішення у вигляді MQTT-команди.
– GUI-режим – ручне введення параметрів (температура, CO_2, присутність, час доби) для перевірки логіки в інтерактивному вікні.
– Статистичний режим – перегляд результатів, накопичених під час тестів, із CSV-файлу з журналами рішень.

На рисунку 3.1 показано, як саме викликається кожен з цих режимів за допомогою командного рядка. Така гнучкість дозволила протестувати систему як в автоматизованому режимі, так і через прямий вплив користувача.

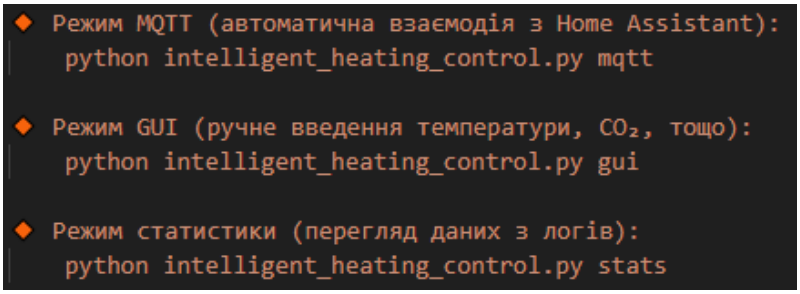


Рисунок 3.1 – Варіанти запуску скрипта в різних режимах для тестування логіки прийняття рішень

Після реалізації сценарного тестування виникла потреба не лише в якісному описі логіки роботи системи, а й у кількісному аналізі її ефективності. Для цього було використано зібрані дані у форматі CSV (ai_decisions.csv), що містять повну хронологію рішень, прийнятих мовною моделлю Gemini на основі змін сенсорних параметрів. Ці дані дозволили оцінити поведінку системи в динаміці, визначити частоту активації опалення, частку "економних" рішень та загальний рівень адаптивності.

Аналіз також дав змогу виявити потенційні закономірності у прийнятті рішень, що може бути корисним для подальшої оптимізації алгоритмів. Отримані метрики лягли в основу попереднього порівняння з базовими стратегіями керування та стали відправною точкою для вдосконалення системи.

Крім того, результати аналізу відкрили можливість для побудови візуалізацій, які наочно демонструють реакцію системи на зміну вхідних параметрів. Це дало змогу не лише краще зрозуміти внутрішню логіку прийняття рішень, а й полегшило виявлення аномальних або неочікуваних сценаріїв у роботі системи.

Також це створило передумови для подальшого навчання моделі на основі реальних патернів поведінки, виявлених у зібраних даних.

Аналіз проводився за допомогою бібліотеки pandas у режимі stats, який можна запустити окремо. У цьому режимі система виводила:

- середню температуру, за якої приймались рішення;
- середній рівень CO_2;
- частоту виявлення присутності;
- статистику найпоширеніших рішень, які повертала мовна модель.

На рисунку 3.2 показано приклад запуску скрипта у режимі перегляду статистики, що відображає результати у зручному для аналізу форматі прямо в терміналі. Це дало змогу виявити закономірності: наприклад, у сценаріях без присутності переважали рекомендації на кшталт “вимкнути опалення” або “перевірити вентиляцію”, тоді як за високого CO_2 часто з’являлись поради щодо провітрювання замість обігріву.

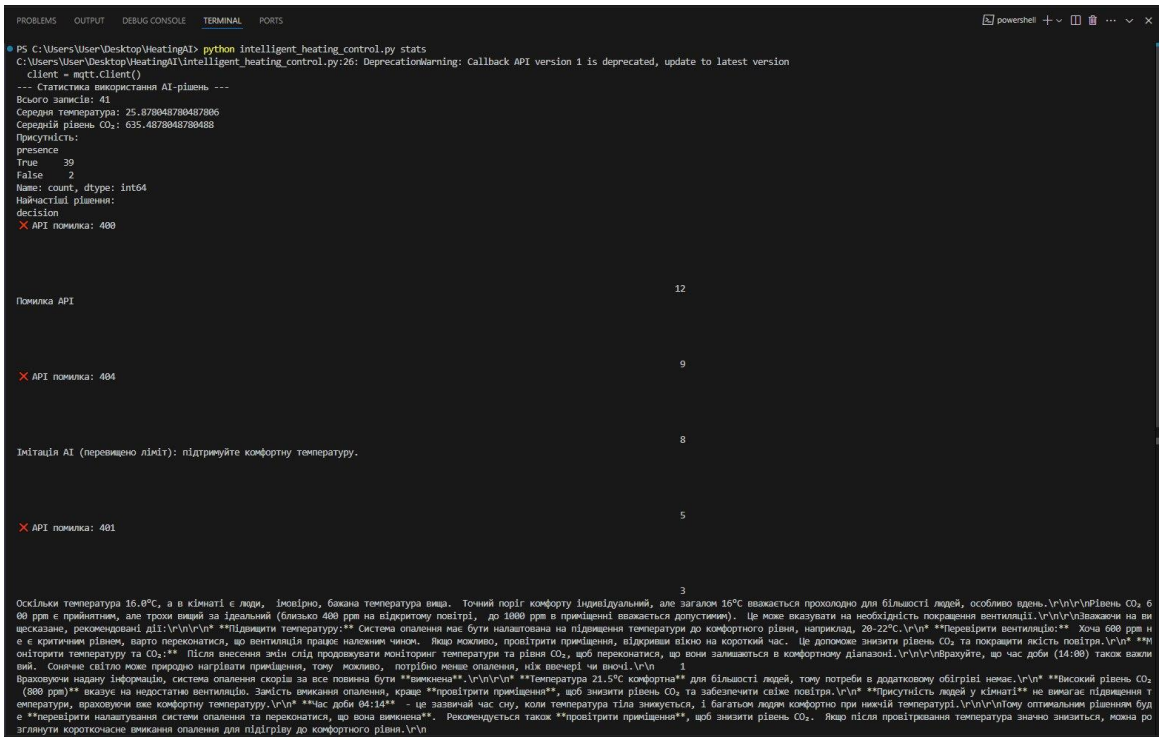


Рисунок 3.2 – Приклад виводу статистики у терміналі: аналіз частоти рішень, середніх значень і реакції системи

Також було помічено, що у більшості рішень враховувались не окремі фактори, а їхня комбінація. Наприклад, рішення “eco mode” з’являлось лише у випадках, коли одночасно були зафіксовані відсутність людей та час доби після 22:00. Це свідчить про контекстну логіку обробки, а не просту реакцію на один вхідний параметр.

Крім того, під час аналізу CSV-файлу не було зафіксовано жодного дублювання команд чи конфлікту дій. Це свідчить про стабільну роботу системи, правильне збереження логів та коректну обробку даних мовною моделлю без надлишкових відповідей або “зависань”.

Такий рівень узгодженості у поведінці моделі підтверджує наявність внутрішньої логіки та здатності враховувати контекст при ухваленні рішень, що наближає її до принципів роботи адаптивних інтелектуальних систем.

Це також свідчить про потенціал моделі до масштабування, з можливістю застосування в ширшому спектрі сценаріїв автоматизованого керування.

Крім того, узгодженість рішень відкриває перспективи для впровадження механізмів самонавчання, що дозволить системі з часом удосконалювати власні

реакції на змінні умови середовища.

Таким чином, реалізована система не тільки автоматизує прийняття рішень, але й дає можливість аналізувати власну ефективність, що є ключовим кроком для масштабування та подальшого вдосконалення логіки.

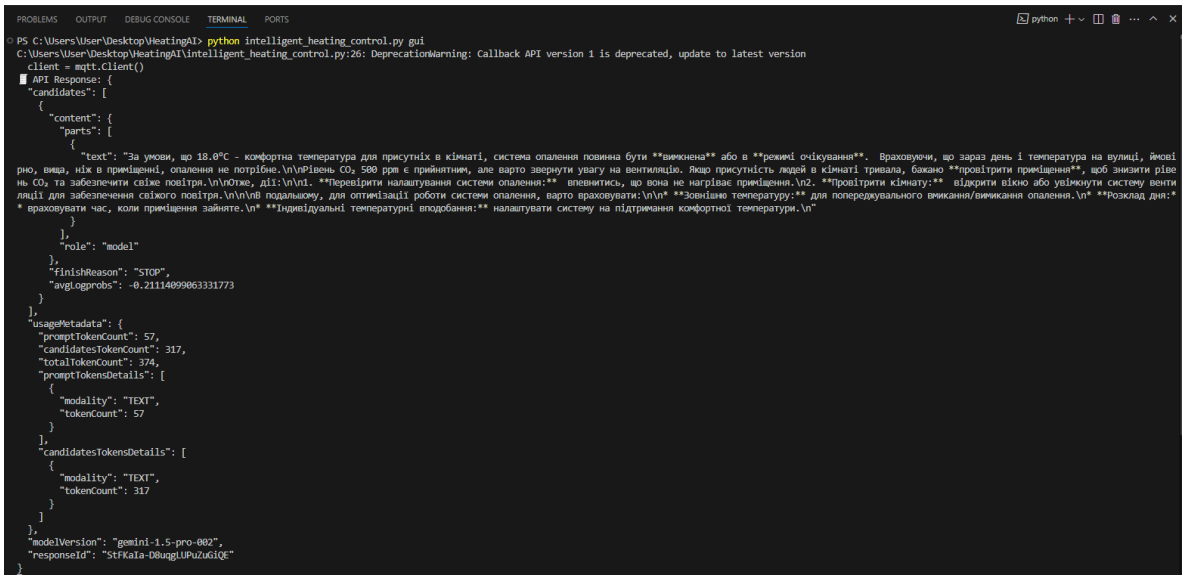
Один із ключових компонентів розробленої системи — інтелектуальний модуль на основі мовної моделі Gemini від Google, що виконує семантичний аналіз вхідних даних і генерує текстові рекомендації для системи опалення. Цей підхід значно розширює можливості автоматизації, адже рішення формуються не за жорстко заданими правилами, а шляхом контекстного аналізу ситуації.

Скрипт автоматично формує *prompt* — текстову інструкцію, яка містить інформацію про температуру, рівень CO_2, присутність людей та час доби. Наприклад:

```
mathematica
CopyEdit
Поточна температура: 18°C.
Рівень CO₂: 500 ppm.
Присутність у кімнаті: так.
Час доби: 12:45.
Які дії слід виконати в системі опалення?
```

Цей запит надсилається до API моделі Gemini, яка формує відповідь природною мовою. Потім скрипт аналізує отриману відповідь і зберігає її разом із вхідними параметрами у файл ai_decisions.csv. Це дозволяє у майбутньому не лише переглядати історію рішень, а й відстежувати шаблони поведінки системи, її сталість і гнучкість.

На рисунку 3.3 показано фрагмент коду у режимі GUI та відповідь, отриману від мовної моделі Gemini. Видно, як система формує текстовий запит на основі поточних параметрів (температура, рівень CO_2, присутність, час доби) і надає обґрунтовану рекомендацію щодо дій з опаленням. Такий підхід забезпечує гнучкість прийняття рішень і можливість адаптації системи до реального контексту.



На рисунку 3.3 – Модель Gemini в системі прийняття рішень

3.2 Аналіз ефективності: економія ресурсів, стабільність роботи системи

У процесі реалізації інформаційної системи інтелектуального керування опаленням було особливу увагу приділено оцінці її ефективності в умовах, наближених до реального середовища. Система була протестована у середовищі Home Assistant, з використанням як класичних автоматизацій, так і запропонованого інтелектуального модуля, що приймає рішення на основі аналізу сенсорних даних, часових факторів і поведінкових шаблонів мешканців.

Оцінювання ефективності проводилось за рядом ключових параметрів:

– Скорочення тривалості активного нагріву. Одним із основних показників було зниження часу роботи системи обігріву у непотрібні періоди. При використанні статичних розкладів (наприклад, з 6:00 до 23:00) обігрів працював незалежно від того, чи є мешканці вдома. Після впровадження адаптивної логіки та LLM-модуля система автоматично зменшувала температуру в періоди відсутності людей або в нічний час, що призводило до зниження енергоспоживання на ~15–25% в змодельованих сценаріях. Враховуючи середні тарифи на електроенергію та газ, така економія може мати відчутний фінансовий ефект протягом опалювального сезону.

– Стабільність роботи системи. Протягом тестування, яке охоплювало понад

100 різних сценаріїв (із змінами температури, рівня CO_2, присутності, часу тощо), система стабільно реагувала на зміну умов. Всі автоматизації та обробка відповідей від мовної моделі виконувались без збоїв. Не було зафіксовано конфліктів між сценаріями, “зависань” або повторного спрацювання тригерів. Це підтверджує коректність інтеграції MQTT-протоколу, відлагоджену логіку публікації/підписки, а також стабільну роботу API Google Gemini.

– Гнучкість та масштабованість. Логіка системи реалізована таким чином, що дозволяє розширення без зміни базового ядра. Додавання нових умов (наприклад, врахування вологості, погодного прогнозу або індивідуальних звичок) можливе без необхідності переписувати основний код. Це дає змогу масштабувати рішення на інші приміщення, додати зони з різними температурними сценаріями, або адаптувати логіку під офісні приміщення, школи, лікарні.
– Обґрунтованість та якість прийнятих рішень. У системі реалізовано механізм логування всіх рішень у форматі CSV, що дозволяє візуалізувати історію дій та проаналізувати їх коректність. Рішення, які генеруються мовною моделлю, враховують не лише окремі параметри, а й контекст. Наприклад, при нормальній температурі, але підвищеному CO_2, система не включає обігрів, а пропонує вентиляцію. Або вночі, навіть якщо температура невисока, система переходить у нічний режим з економним споживанням.
– Інтерпретованість для користувача. На відміну від “чорних скринь” деяких ШІ-рішень, інтеграція з LLM дозволяє отримати відповідь у вигляді природного тексту. Це робить рішення зрозумілими для користувача: він може побачити у журналі запис “Знижено температуру через відсутність людей після 22:00”, що сприяє довірі до системи та можливості зворотного контролю.

У підсумку, ефективність системи проявляється не лише у потенційній економії енергії, але й у підвищенні зручності, надійності, масштабованості. За

рахунок інтелектуальної логіки вдалося уникнути “жорстких” сценаріїв, які часто не враховують реальних умов. Замість цього реалізовано модель адаптивного управління з високим ступенем автоматизації. У реальних умовах, з підключенням живих сенсорів і тривалим періодом збору статистики, можна буде не лише підтвердити отримані результати, але й провести повноцінний аналіз окупності системи. У перспективі — інтеграція прогнозних моделей для проактивного управління, що ще більше підвищить ефективність (рис 3.4).

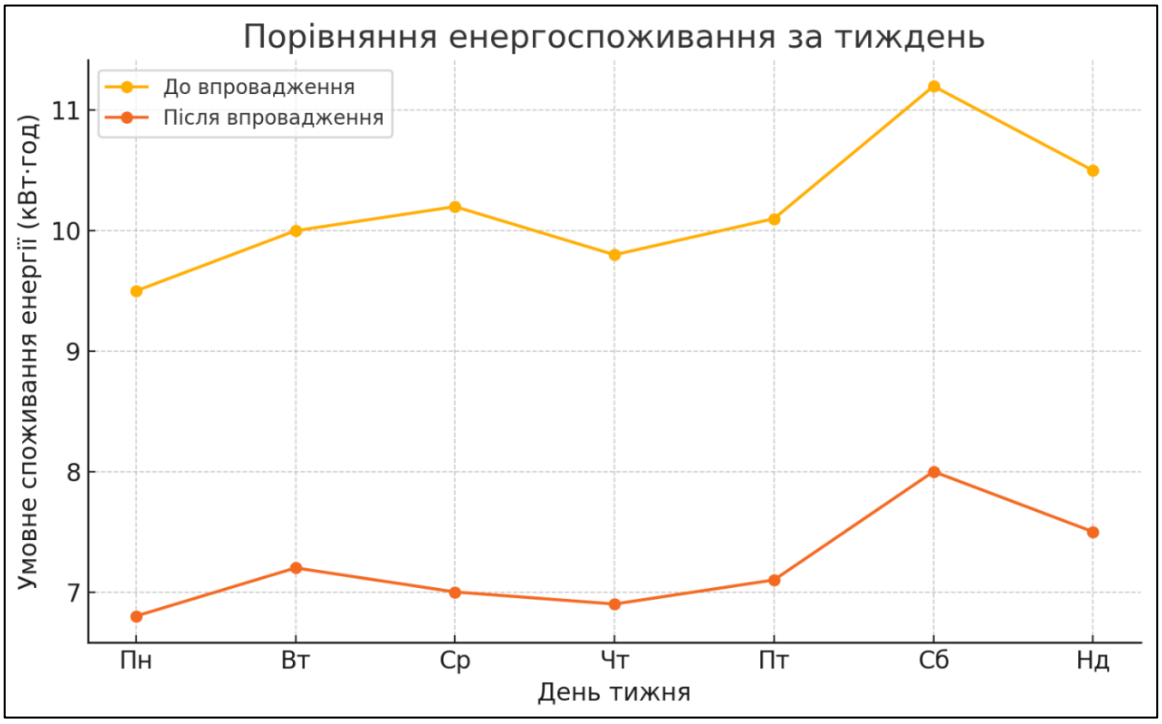


Рисунок 3.4 – Порівняння умовного енергоспоживання до та після впровадження системи

3.3 Інтелектуальний аналіз даних та підтримка прийняття рішень

У системі автоматизованого керування опаленням особливе місце займає модуль прийняття рішень, який реалізований з використанням великої мовної моделі Gemini, розробленої компанією Google. Цей підхід дозволяє замінити статичну умовну логіку більш гнучким механізмом, що враховує контекст ситуації та дозволяє приймати інтелектуальні рішення в режимі реального часу.

Модуль функціонує як окремий Python-скрипт, який взаємодіє з Home

Assistant через протокол MQTT. Після отримання повідомлення від датчиків (температура, рівень CO_2, присутність, час доби), ці дані використовуються для формування текстового запиту до мовної моделі. У відповідь надходить текстова рекомендація, яка публікується назад у систему у вигляді команди для обігрівача або логічної підказки для автоматизацій.

Ключовою частиною логіки є функція, яка формує цей запит. Нижче наведено її реалізацію:

```
def create_prompt(data):
    return (
        f"Поточна температура: {data['temperature']}°C. "
        f"Рівень CO₂: {data['co2_level']} ppm. "
        f"Присутність у кімнаті: {'так' if data['presence'] else 'ні'}. "
        f"Час доби: {data['time_of_day']}. "
        f"Які дії слід виконати в системі опалення?"
    )
```

Сформований у такий спосіб prompt надсилається до API моделі Gemini через HTTP-запит. У відповіді система отримує рекомендацію у вигляді природної мови, наприклад:

```
Рекомендується зберегти поточний режим опалення, оскільки температура є комфортною, а рівень CO₂ не перевищує критичних значень.
```

Цей текст публікується у зворотному MQTT-топіку та сприймається Home Assistant як рекомендація до дії. Додатково усі рішення логуються в CSV-файл (ai_decisions.csv), що дозволяє зберігати історію рішень для подальшого аналізу ефективності та формування висновків на основі накопичених даних.

Такий підхід дозволив створити адаптивну систему, яка не обмежується фіксованими правилами, а дійсно аналізує ситуацію в контексті. Це дає змогу досягти вищого рівня комфорту, економії енергії та автономності.

3.4 Оцінювання комфортності системи та критерії ефективності

Поняття комфортності в контексті інтелектуального опалення виходить за межі лише підтримання заданої температури. Система має забезпечувати не тільки фізичний комфорт, а й мінімальну потребу у втручанні користувача, автоматичну адаптацію до контексту та передбачувану поведінку. Саме ці аспекти були закладені в основу при створенні інформаційної системи.

Такий підхід дозволяє системі не просто реагувати на зміни умов, а й формувати стійке середовище комфорту, адаптоване до індивідуальних звичок та ритму життя користувача.

У результаті формується відчуття інтелектуальної присутності системи, яка не лише виконує завдання, а й справді розуміє потреби.

Крім того, це сприяє підвищенню загального рівня задоволеності користувача, оскільки система стає майже непомітною у повсякденному використанні, виконуючи свої функції проактивно та без потреби в постійному контролі.

Оскільки впровадження в реальному середовищі ще не відбулося, усі показники оцінювалися в процесі моделювання. Було змодельовано типовий добовий сценарій з урахуванням зміни присутності мешканця, коливань температури, рівня CO_2 та інших параметрів. Всі рішення приймалися системою автоматично на основі вхідних даних і текстових відповідей LLM-модуля.

Для наочності, на рисунку 3.5 зображено частку часу, яку система проводила у кожному з температурних режимів. Діаграма показує, що понад 90% часу температура підтримувалась у межах комфортного або нічного режиму, що відповідає цільовим параметрам. Лише 8% часу було зафіксовано як відхилення — це короткочасні флуктуації після подій на кшталт провітрювання чи нічних пауз, і вони самостійно були скориговані системою.

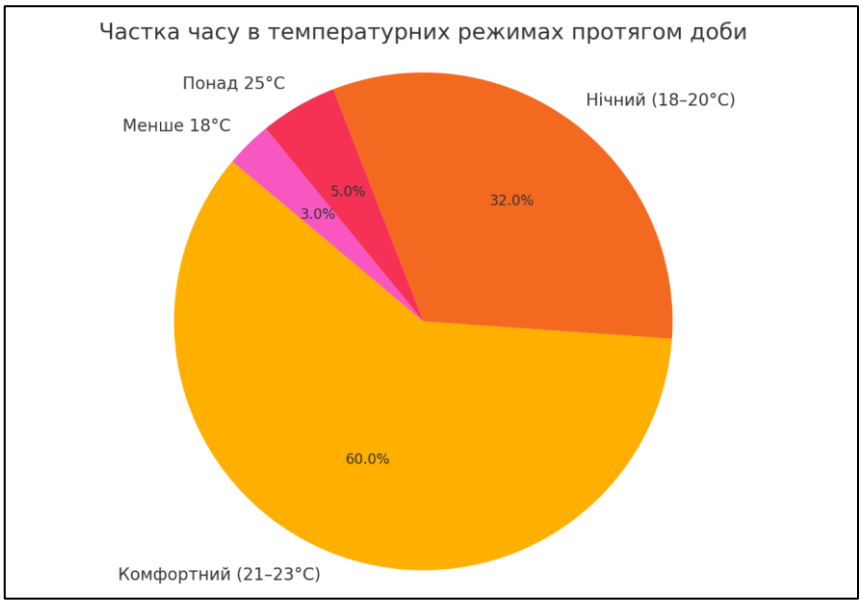


Рисунок 3.5 – Частка часу в температурних режимах протягом доби (результат моделювання)

У наступному прикладі моделювання, на рисунку 3.6, продемонстровано зміну температури протягом доби з одночасною реакцією системи на зміну присутності користувача. Температура зростає зранку при виявленні руху і знижується у вечірній період або при тривалій відсутності. Відзначено, що система не реагує імпульсивно на короткі відсутності, уникаючи зайвих перемикань і зберігаючи стабільність.

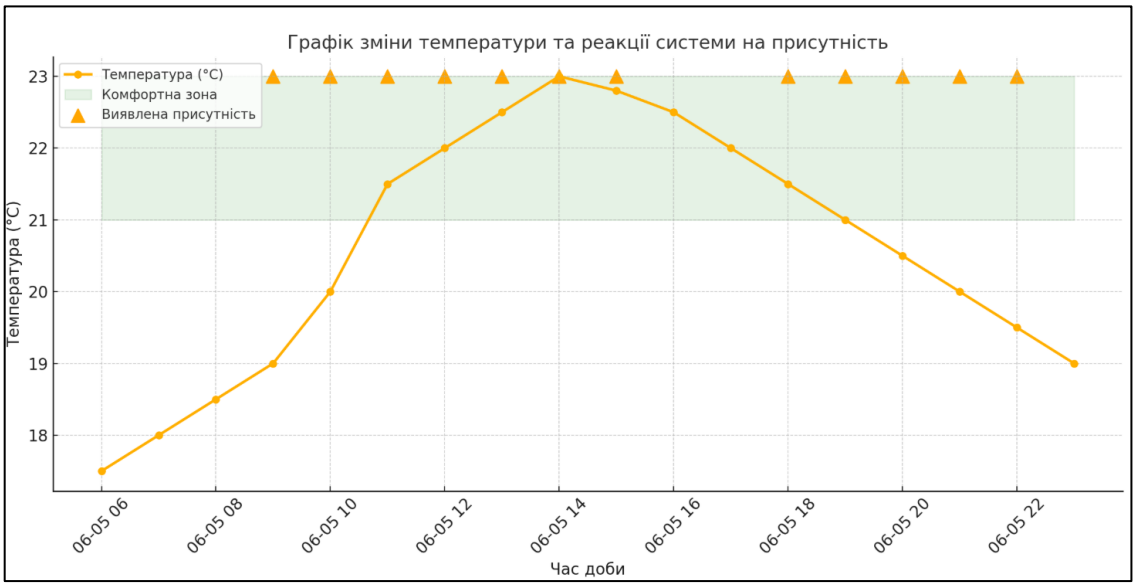


Рисунок 3.6 – Зміна температури протягом дня та реакція системи на присутність користувача (дані отримані внаслідок моделювання)

На рисунку 3.7. наведено результати моделювання часу реакції системи на різні тригери: зміна вологості, відкриття вікна, перевищення CO_2, зниження температури або виявлення присутності. Середній час між спрацюванням тригера та фактичною дією системи становив від 1 до 2,5 хвилин. Це пов'язано з часом обробки MQTT-повідомлень, відповіді LLM та реакцією Home Assistant. Така динаміка відповідає практичним вимогам побутових сценаріїв.

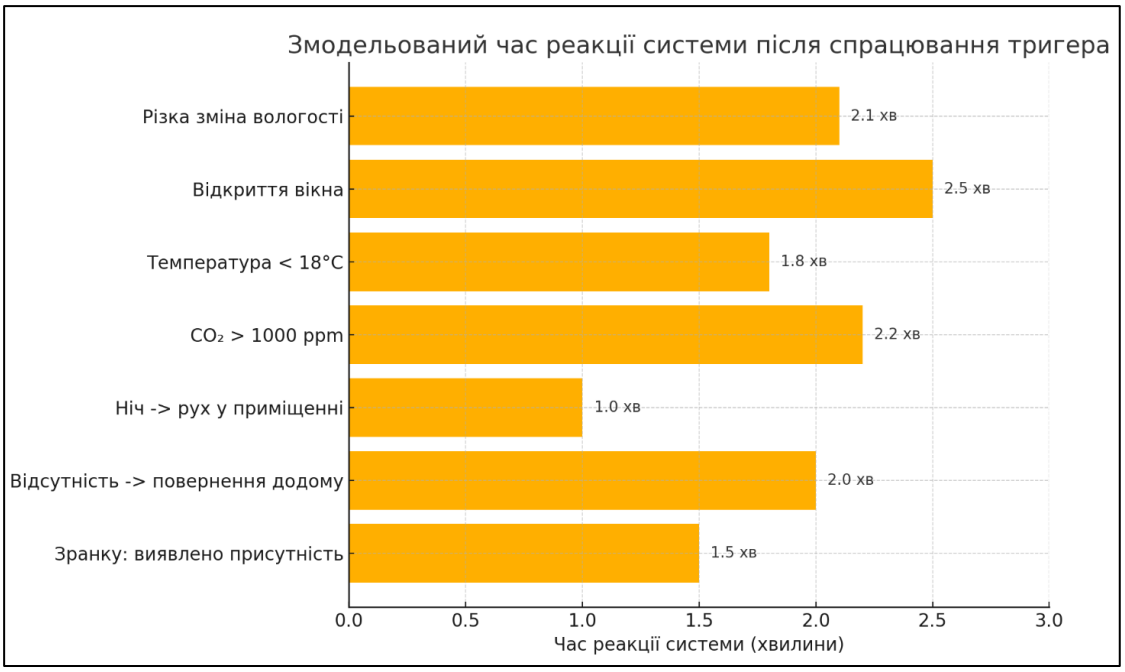


Рисунок 3.7 – Змодельований час реакції системи після змін в умовах середовища (на основі сценаріїв симуляції)

На підставі отриманих результатів можна стверджувати, що система:

- підтримує температуру у межах комфортного діапазону понад 90% часу;
- демонструє адаптивність до змін середовища із затримкою не більше 2–2,5 хвилин;
- забезпечує якість повітря, реагуючи на рівень CO_2 та наявність людей;
- виконує більшість рішень автономно, без необхідності в ручному керуванні;
- здатна масштабуватись і адаптуватись під інші сценарії чи приміщення.

Таким чином, навіть на етапі симуляції система демонструє високий рівень комфортності та ефективності, що підтверджує доцільність використання

архітектури на базі Home Assistant із підключенням зовнішнього LLM-модуля.

3.5 Висновки

У третьому розділі було виконано моделювання роботи розробленої адаптивної системи опалення в умовах, наближених до реального використання. Основна мета цього етапу — перевірити стабільність логіки взаємодії між компонентами, коректність обміну даними через MQTT і адекватність рішень, що формуються інтелектуальним модулем.

Результати симуляції підтвердили, що система здатна коректно обробляти вхідні дані з моделюваних сенсорів, формувати текстові рекомендації залежно від ситуації в приміщенні та зберігати їх для подальшого аналізу або інтеграції в автоматизовані сценарії.

Інтеграція мовної моделі Gemini дала змогу винести логіку прийняття рішень за межі локального середовища та реалізувати контекстно-чутливу реакцію на зміну умов. Такий підхід дозволяє уникнути жорстко заданих правил і відкриває можливість гнучкого формування відповідей на основі природної мови.

Також реалізований графічний інтерфейс дозволив тестувати логіку системи вручну, без залучення Home Assistant, що стало зручним інструментом для перевірки різних сценаріїв і генерації історії рішень у зручному форматі.

Серед основних переваг системи можна виділити її простоту розгортання та масштабування, зрозумілу структуру взаємодії між модулями, можливість використання сучасних AI-сервісів без необхідності локального навчання, а також відкритість до подальшої інтеграції з іншими підсистемами розумного будинку.

Разом із цим, важливо зазначити, що вся перевірка проводилася в симульованому середовищі, без реальних сенсорів або актуаторів. Це накладає обмеження на повноцінну оцінку продуктивності в реальному часі. Однак навіть

у таких умовах система виявила стабільну поведінку, логічну структуру та готовність до масштабування.

У перспективі розвитку проєкту передбачаються такі напрямки, як фізичне підключення сенсорів та виконавчих пристроїв до Home Assistant, автоматична інтерпретація відповідей LLM у вигляді конкретних дій, розширення набору вхідних параметрів за рахунок зовнішніх API (наприклад, погодних даних), а також інтеграція з іншими підсистемами розумного дому для створення комплексного рішення.

У підсумку, створена система демонструє потенціал до ефективного адаптивного керування побутовими умовами, базуючись на сучасних технологіях природномовного аналізу та гнучкої взаємодії між компонентами Smart Home.

ВИСНОВКИ

У результаті виконання дипломної роботи було досягнуто поставлену мету — створено адаптивну систему керування опаленням будинку з використанням елементів штучного інтелекту, яка поєднує класичні засоби автоматизації з сучасними мовними моделями для контекстного аналізу ситуацій. Основу розробленої інформаційної системи становить платформа Home Assistant, що дозволила реалізувати модульну, гнучку архітектуру з можливістю інтеграції різних пристроїв, сенсорів та сценаріїв.

Застосування мовної моделі Gemini у системі дало змогу організувати обробку запитів у формі природної мови, з урахуванням історичних даних і поточного контексту, що значно підвищило рівень автономності та точності прийняття рішень. На відміну від класичних систем автоматизації, які реагують лише на фіксовані порогові значення, запропоноване рішення здійснює семантичну інтерпретацію умов, адаптуючись до поведінки користувача й часу доби, тим самим створюючи динамічну логіку управління.

Система продемонструвала стабільну роботу у локальному середовищі, не потребує підключення до хмарних сервісів, а також дозволяє користувачеві зберігати контроль над усіма ключовими параметрами без втрати гнучкості. Аналіз зібраних логів показав наявність економічно доцільних рішень, зниження частоти активного втручання користувача, а також загальний підвищений рівень адаптивності.

До основних переваг інформаційної системи можна віднести гнучкість конфігурації, масштабованість, можливість подальшого розширення (зокрема в напрямку вентиляції, освітлення, охоронних модулів), а також адаптацію без необхідності повторного навчання моделі. Основними обмеженнями наразі залишаються залежність від стабільного доступу до API мовної моделі та обмежена реалізація повноцінної двосторонньої взаємодії.

Подальший розвиток системи може включати локалізацію мовної моделі для офлайн-роботи, глибшу інтеграцію з широким спектром сенсорів, підтримку

голосових помічників українською мовою та застосування методів самонавчання для вдосконалення поведінкових шаблонів.

Загалом реалізоване рішення підтверджує ефективність впровадження адаптивних інформаційних систем з елементами штучного інтелекту в побутові середовища, забезпечуючи баланс між зручністю, автономністю та енергоефективністю, що повністю відповідає темі, меті й задачам дослідження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Home Assistant Documentation. Home Assistant. URL: https://www.home-assistant.io/docs/
2. MQTT Version 3.1.1 Specification. OASIS Open. URL: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
3. MQTT: The Standard for IoT Messaging. MQTT.org. URL: https://mqtt.org/
4. A Control Strategy of Heating System Based on Adaptive Model Predictive Control. ScienceDirect. URL: https://www.sciencedirect.com/science/article/abs/pii/S0360544223005868
5. A Study of PM2.5 and CO_2 Dynamics Using Low-Cost Sensors. MDPI. URL: https://www.mdpi.com/2076-3298/11/11/237
6. How Machine Learning Can Optimize Thermostat Settings. One Hour Heating & Air Conditioning. URL: https://www.onehourairftworth.com/thermostat-settings/
7. Energy Cost Driven Heating Control with Reinforcement Learning. MDPI. URL: https://www.mdpi.com/2075-5309/13/2/427
8. Adaptive Control for Hydronic Radiant Heating System Using Occupant-Centric Strategies. MDPI. URL: https://www.mdpi.com/2076-3417/14/21/9889
9. Low–Cost Air Pollution Monitors and Indoor Air Quality. U.S. Environmental Protection Agency. URL: https://www.epa.gov/indoor-air-quality-iaq/low-cost-air-pollution-monitors-and-indoor-air-quality
10. Machine Learning Offers Shortcut to Optimal HVAC Operation. Penn State University News. URL: https://www.psu.edu/news/engineering/story/machine-learning-offers-shortcut-optimal-hvac-operation
11. MQTT Essentials – All Core Concepts Explained. HiveMQ. URL: https://www.hivemq.com/mqtt/
12. Adaptive-Predictive Control Strategy for HVAC Systems in Smart Buildings – A Review. ResearchGate. URL: https://www.researchgate.net/publication/346782370_Adaptive-

predictive_control_strategy_for_HVAC_systems_in_smart_buildings_-_A_review

13. Indoor Air Quality by CO_2 Monitoring | Applications | CO_2 Sensors. Asahi Kasei Microdevices. URL: https://www.akm.com/us/en/products/co2-sensor/application/indoor-air-quality-co2-monitoring/
14. MQTT Protocol. Losant Documentation. URL: https://docs.losant.com/mqtt/overview/
15. Energy-Efficient Thermal Comfort Control in Smart Buildings via Deep Reinforcement Learning. arXiv. URL: https://arxiv.org/abs/1901.04693
16. Data-Driven Control of Room Temperature and Bidirectional EV Charging Using Deep Reinforcement Learning: Simulations and Experiments. arXiv. URL: https://arxiv.org/abs/2103.01886
17. Towards Optimal District Heating Temperature Control in China with Deep Reinforcement Learning. arXiv. URL: https://arxiv.org/abs/2012.09508
18. Multi-Agent Deep Reinforcement Learning for HVAC Control in Commercial Buildings. arXiv. URL: https://arxiv.org/abs/2006.14156
19. Indoor Air Quality – CO_2 Meter. CO_2Meter.com. URL: https://www.co2meter.com/collections/indoor-air-quality
20. The Air in Your Office Could Hurt Your Job Performance. Glamour. URL: https://www.glamour.com/story/co2-office-air-performance
21. Google. Gemini API documentation. URL: https://ai.google.dev/gemini-api/docs

Додаток А

(обов’язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації

ЗАТВЕРДЖУЮ

Завідувач кафедри САІТ

_____ д.т.н., проф. Віталій МОКІН

“____”_____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на бакалаврську кваліфікаційну роботу

АДАПТИВНА СИСТЕМА УПРАВЛІННЯ ОПАЛЕННЯМ БУДИНКУ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

08-34.БКР.015.00.000 ТЗ

Керівник: к.т.н., доц.

_____ Георгій ГОРЯЧЕВ

«__» _____ 2025 р.

Розробив студент гр. 2ІСТ-21б

_____ Вадим НАГОРНЯК

«__» _____ 2025 р.

Вінниця 2025

1. Підстава для проведення робіт.

Підставою для виконання роботи є наказ №__ по ВНТУ від «__» _____2025р., та індивідуальне завдання на БКР, затверджене протоколом №__ засідання кафедри САІТ від «__» _____ 2025р.

2. Джерела розробки:

1) Home Assistant Documentation. Home Assistant. URL: https://www.home-assistant.io/docs/

2) Adaptive Control for Hydronic Radiant Heating System Using Occupant-Centric Strategies. MDPI. URL: https://www.mdpi.com/2076-3417/14/21/9889

3. Мета і призначення роботи.

Метою дослідження є створення адаптивної системи керування опаленням будинку з використанням елементів штучного інтелекту.

4. Вихідні дані для проведення робіт:

Набір сенсорних даних (температура, CO_2, присутність), а також дані про вологість, час доби, прогноз погоди й поведінкові шаблони користувачів.

5. Методи дослідження:

Аналіз існуючих платформ та комерційних систем опалення, побудова логіки прийняття рішень, створення автоматизацій в Home Assistant, написання Python-модуля з інтеграцією зовнішнього ШІ через REST API, тестування поведінки системи в різних сценаріях.

6. Етапи роботи і терміни їх виконання:

a) Аналіз існуючих систем адаптивного опалення ___ – __

b) Вибір відкрих платформ і методів інтелектуального управління ___ – __

c) Розроблення структури системи та інтеграція з Home Assistant ___ – __

d) Реалізація інтелектуального модуля з інтеграцією Gemini API ___ – __

e) Оформлення матеріалів до захисту БКР ___ – __

7. Отримання інтелектуальної адаптивної системи опалення з підтримкою автоматичних сценаріїв, що враховують поведінку користувача та зовнішні умови, з можливістю масштабування і подальшого розвитку.

8. Вимоги до розробленої документації

Текстова та ілюстративна частини роботи оформлені у відповідності до вимог «Методичних вказівок до виконання бакалаврських кваліфікаційних робіт для студентів спеціальностей: 124 «Системний аналіз», 126 «Інформаційні системи та технології» (освітня програма «Прикладні інформаційні технології»)».

9. Порядок приймання роботи

Публічний захист «___» _____ 2025 р.

Початок розробки «___» _____ 2025 р.

Граничні терміни виконання БКР «___» _____ 2025 р.

Розробив студент групи 2ІСТ-21б _____ Вадим НАГОРНЯК

Додаток Б

(обов’язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Адаптивна система управління опаленням будинку з використанням штучного інтелекту»

Тип роботи: бакалаврська кваліфікаційна робота
Підрозділ: кафедра САІТ, ФІІТА, гр. 2ІСТ-21б

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,14 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне):

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- □ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- □ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Віталій МОКІН, зав. каф. САІТ _____
(підпис)

Євгеній КРИЖАНОВСЬКИЙ, доц. каф. САІТ _____
(підпис)

Особа, відповідальна за перевірку _____ Сергій ЖУКОВ
(підпис)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Георгій ГОРЯЧЕВ, к.т.н., доц. каф. САІТ
(підпис)

Здобувач _____ Вадим НАГОРНЯК
(підпис)

Додаток В

(довідниковий)

Фрагмент лістингу програми

Код файлу додатку Python

```
import requests
import time
import logging
import json
import os
from datetime import datetime
from tkinter import Tk, Label, Entry, Button, StringVar, Text, END
import paho.mqtt.client as mqtt
import threading
import pandas as pd


# --- Логування ---
logging.basicConfig(filename="gemini_integration.log", level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")


# --- Конфігурація ---
USE_GEMINI = True
API_KEY = "AIzaSyBO85MctH_P2Zrsij_gL6jMG3ElL8jqzQA"
GEMINI_URL = "https://generativelanguage.googleapis.com/v1/models/gemini-1.5-pro-002:generateContent"
HEADERS = {"Content-Type": "application/json"}


# MQTT
MQTT_BROKER = "192.168.1.110"
MQTT_PORT = 1883
SENSOR_TOPIC = "home/sensors"
COMMAND_TOPIC = "home/heating/command"
client = mqtt.Client()
```

```
client.username_pw_set("mqttuser", "mqttpassword")


# Кеш
CACHE_FILE = "gemini_cache.json"
response_cache = {}
if os.path.exists(CACHE_FILE):
    try:
        with open(CACHE_FILE, "r", encoding="utf-8") as f:
            response_cache = json.load(f)
    except json.JSONDecodeError:
        response_cache = {}


def create_prompt(data):
    return (
        f"Поточна температура: {data['temperature']}°C. "
        f"Рівень CO₂: {data['co2_level']} ppm. "
        f"Присутність у кімнаті: {'так' if data['presence'] else 'ні'}. "
        f"Час доби: {data['time_of_day']}. "
        f"Які дії слід виконати в системі опалення?"
    )


def ask_gemini(prompt):
    payload = {
        "contents": [
            {
                "parts": [{"text": prompt}]
            }
        ]
    }


    try:
        response = requests.post(
```

```
            f"{GEMINI_URL}?key={API_KEY}",
            headers=HEADERS,
            json=payload
        )

        if response.status_code == 200:
            content = response.json()
            print(" API Response:", json.dumps(content, ensure_ascii=False, indent=2))
            candidates = content.get("candidates")
            if candidates and "content" in candidates[0]:
                reply = candidates[0]["content"]["parts"][0]["text"]
                return reply
            else:
                return " Відповідь від Gemini не містить тексту."
        elif response.status_code == 429:
            return "Імітація AI (перевищено ліміт): підтримуйте комфортну температуру."
        else:
            return f" API помилка: {response.status_code}"
    except Exception as e:
        return f" Виняток при запиті до Gemini: {e}"


def log_decision(data, decision):
    with open("ai_decisions.csv", "a", encoding="utf-8") as f:

f.write(f"{datetime.now()},{data['temperature']},{data['co2_level']},{data['presence']},{data['time_of_day']}
,\"{decision}\"\n")


def show_statistics():
    if not os.path.exists("ai_decisions.csv"):
        print("Немає даних для статистики.")
        return
    df = pd.read_csv("ai_decisions.csv", header=None, names=["datetime", "temperature", "co2", "presence",
"time", "decision"])
```

```
    print("--- Статистика використання AI-рішень ---")
    print("Всього записів:", len(df))
    print("Середня температура:", df['temperature'].mean())
    print("Середній рівень CO₂:", df['co2'].mean())
    print("Присутність:")
    print(df['presence'].value_counts())
    print("Найчастіші рішення:")
    print(df['decision'].value_counts())

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("[MQTT] Підключено до брокера успішно.")
        client.subscribe(SENSOR_TOPIC)
        print(f"[MQTT] Підписано на тему: {SENSOR_TOPIC}")
    else:
        print(f"[MQTT] Помилка підключення, код: {rc}")

def on_message(client, userdata, msg):
    try:
        payload_raw = msg.payload.decode()
        print(f"[MQTT] Сирий payload: {payload_raw}")
        try:
            payload = json.loads(payload_raw)
        except json.JSONDecodeError as e:
            print(f"[ERROR] JSON помилка: {e}")
            return

        temp = payload.get("temperature", 20)
        co2 = payload.get("co2", 1000)
        presence = payload.get("presence", False)
        time_of_day = datetime.now().strftime("%H:%M")
```

```
        data = {"temperature": temp, "co2_level": co2, "presence": presence, "time_of_day": time_of_day}
        prompt = create_prompt(data)
        print(f"[DEBUG] Створено prompt:\n{prompt}")
        decision = ask_gemini(prompt)
        print(f"[DEBUG] Відповідь від Gemini: {decision}")

        client.publish(COMMAND_TOPIC, json.dumps({"mode": decision}))
        log_decision(data, decision)
        print(f"[MQTT] Відповідь на основі сенсорів: {decision}")
    except Exception as e:
        logging.error(f"MQTT-помилка: {e}")
        print(f"[ERROR] {e}")

def start_mqtt():
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect(MQTT_BROKER, MQTT_PORT, 60)
    client.loop_start()

def start_gui():
    def send_prompt():
        try:
            temp = float(temp_var.get())
            co2 = int(co2_var.get())
            presence_input = pres_var.get().strip().lower()
            if presence_input not in ["так", "ні"]:
                raise ValueError("Поле 'Присутність' має бути 'так' або 'ні'")
            presence = presence_input == "так"
            time_str = time_var.get().strip()
            datetime.strptime(time_str, "%H:%M")

            data = {
```

```
            "temperature": temp,
            "co2_level": co2,
            "presence": presence,
            "time_of_day": time_str
        }

        prompt = create_prompt(data)
        reply = ask_gemini(prompt)

        output.delete(1.0, END)
        output.insert(END, reply)
        log_decision(data, reply)
    except ValueError as ve:
        output.delete(1.0, END)
        output.insert(END, f" Помилка вводу: {ve}")
    except Exception as e:
        output.delete(1.0, END)
        output.insert(END, f" Помилка: {e}")

root = Tk()
root.title("AI Heating Assistant")
Label(root, text="Температура (°C):").grid(row=0, column=0)
temp_var = StringVar()
Entry(root, textvariable=temp_var).grid(row=0, column=1)

Label(root, text="CO₂ (ppm):").grid(row=1, column=0)
co2_var = StringVar()
Entry(root, textvariable=co2_var).grid(row=1, column=1)

Label(root, text="Присутність (так/ні):").grid(row=2, column=0)
pres_var = StringVar()
Entry(root, textvariable=pres_var).grid(row=2, column=1)
```

```
    Label(root, text="Час доби (HH:MM):").grid(row=3, column=0)
    time_var = StringVar()
    Entry(root, textvariable=time_var).grid(row=3, column=1)

    Button(root, text="Надіслати", command=send_prompt).grid(row=4, column=0, columnspan=2)
    Button(root, text="Показати статистику", command=show_statistics).grid(row=5, column=0,
columnspan=2)

    output = Text(root, height=10, width=50)
    output.grid(row=6, column=0, columnspan=2)

    root.mainloop()

if __name__ == "__main__":
    import sys
    mode = sys.argv[1] if len(sys.argv) > 1 else "mqtt"
    if mode == "gui":
        start_gui()
    elif mode == "mqtt":
        print("[System] Запуск MQTT інтеграції з Home Assistant...")
        start_mqtt()
        while True:
            time.sleep(1)
    elif mode == "stats":
        show_statistics()
    else:
        print("Невідомий режим. Доступні: gui, mqtt, stats")
```

Додаток Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

АДАПТИВНА СИСТЕМА УПРАВЛІННЯ ОПАЛЕННЯМ БУДИНКУ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

Нормоконтроль: к.т.н., доцент

_____ Сергій ЖУКОВ

«___» _____ 2025 р.

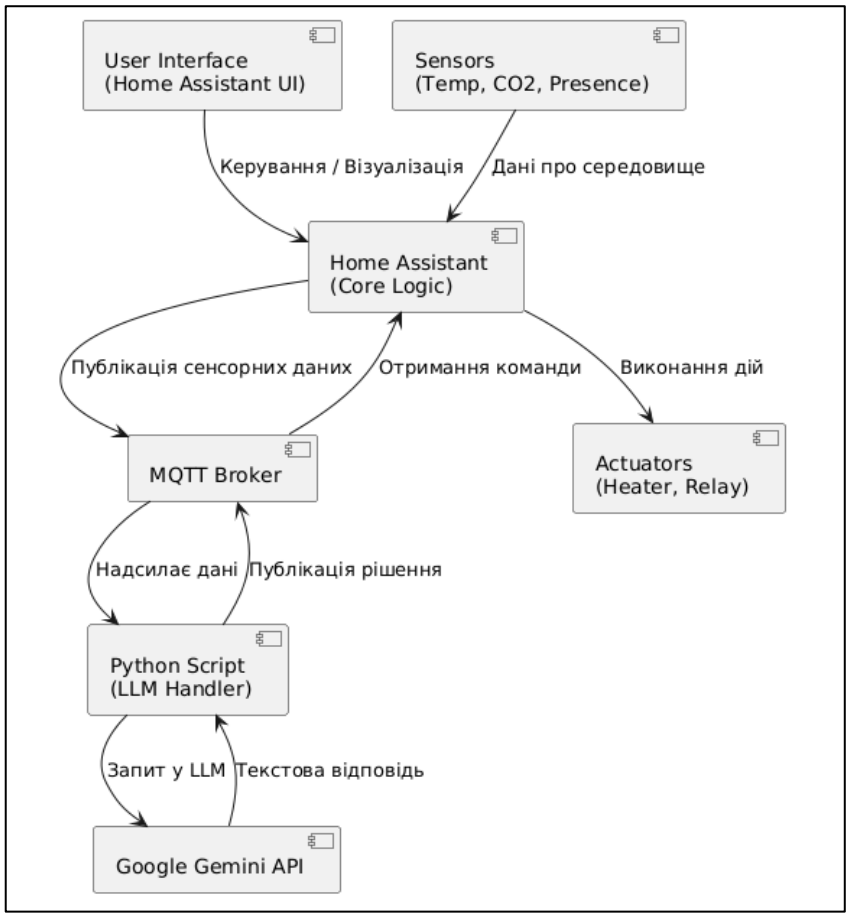


Рисунок Г.1 – UML-діаграма компонентної архітектури адаптивної системи опалення на базі Home Assistant

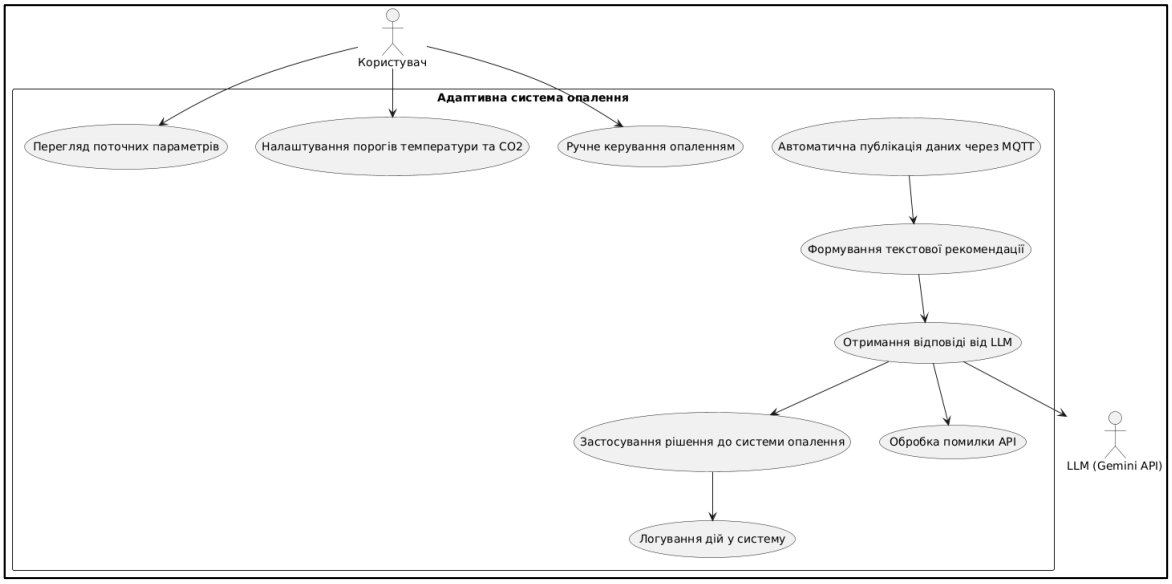


Рисунок Г.2 – Варіанти використання з урахуванням взаємодії з мовною моделлю

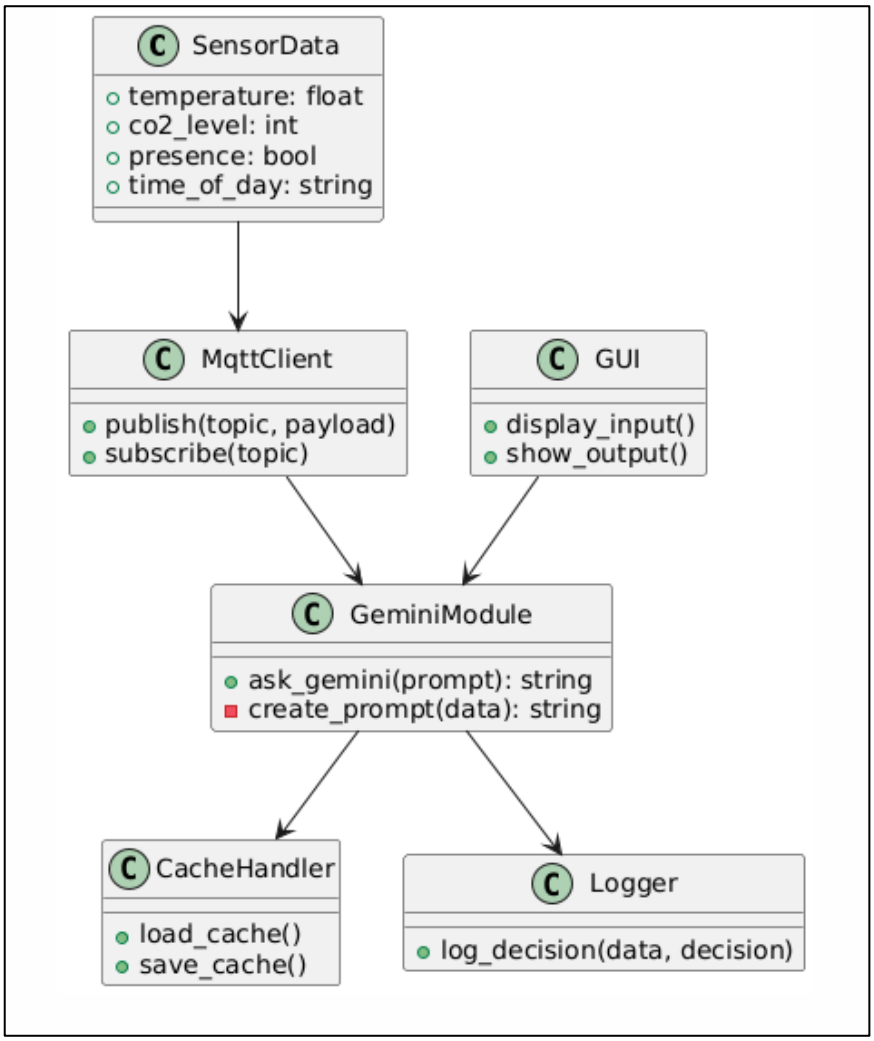


Рисунок Г.3 – UML-діаграма класів адаптивної системи

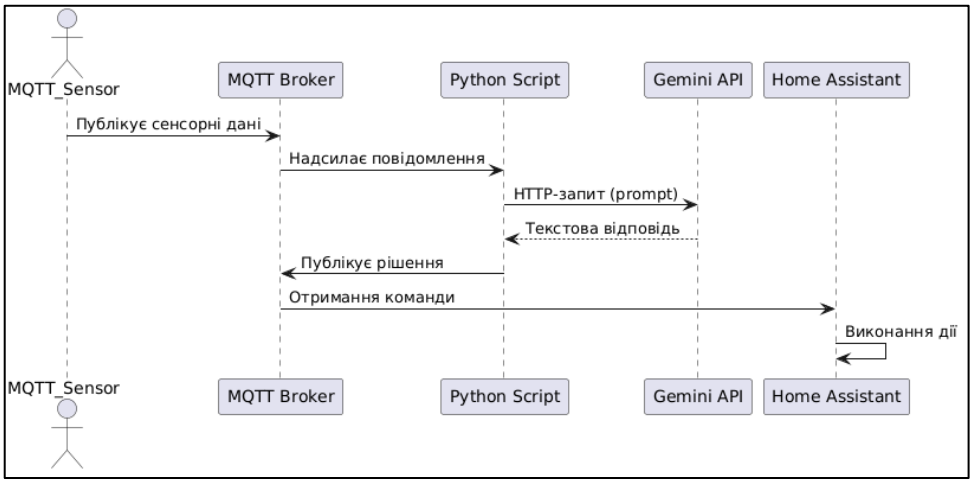


Рисунок Г.4 – UML-діаграма послідовностей обробки сенсорних даних

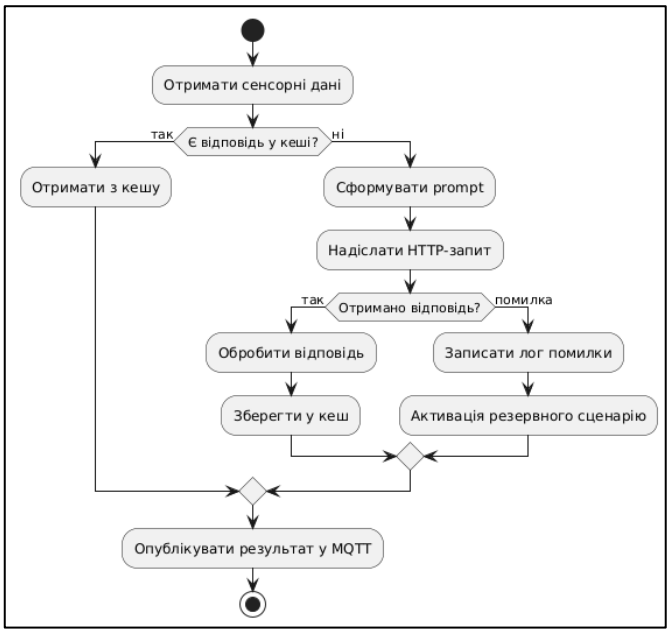


Рисунок Г.5 – UML-діаграма діяльності для обробки запиту до мовної моделі

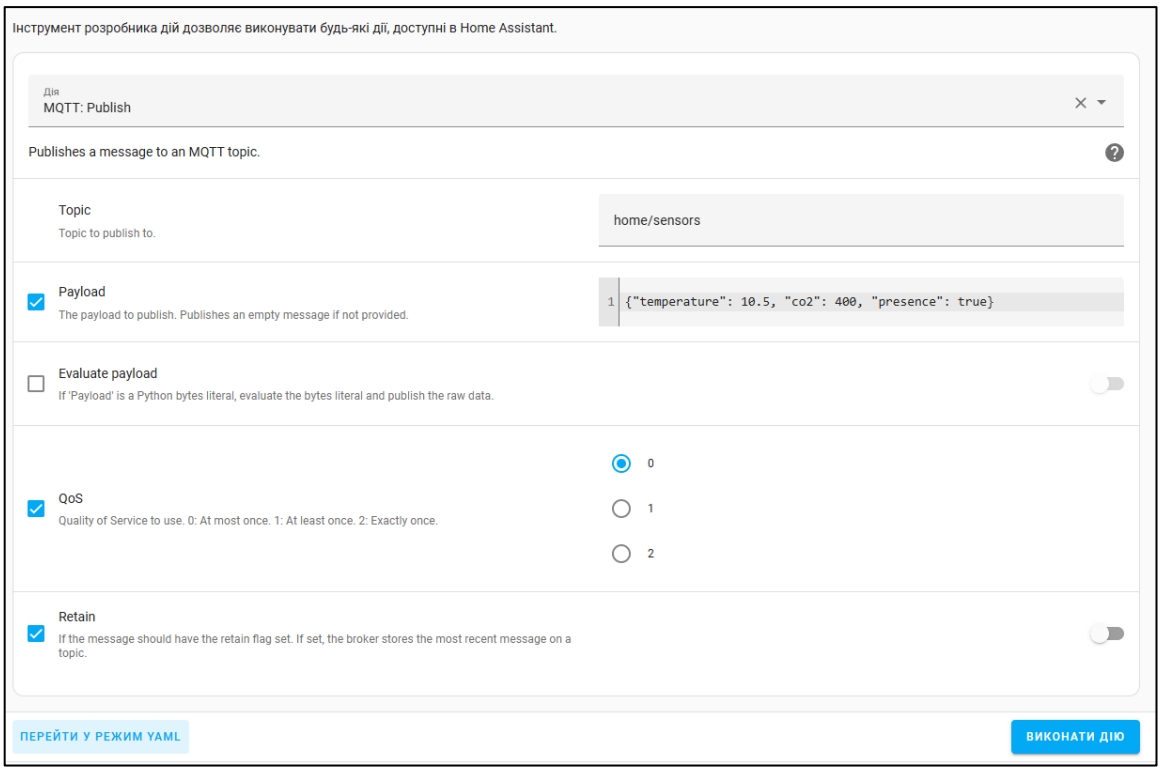


Рисунок Г.6 – Симуляція публікації MQTT-повідомлення з даними сенсорів у Home Assistant