ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Методи штучного інтелекту в розробці домашньої "розумної" екосистеми»

на здобуття освітнього ступеня магістра

зі спеціальності 124 Системний аналіз

освітньо-професійної програми «Інтелектуальні системи управління»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ _____

(підпис) Ім’я, ПРІЗВИЩЕ здобувача

Виконав: здобувач вищої освіти гр. САДМ-61

Дмитро СИДОРЕНКО

Керівник: Ровіл НАФЄЄВ

Рецензент: _____

Київ-2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра: Інформаційних систем та мереж

Ступінь вищої освіти: Магістр

Спеціальність: 124 «Системний аналіз»

Освітньо-професійна програма: Інтелектуальні системи управління

ЗАТВЕРДЖУЮ
Завідувач кафедри ІСТ

_____ Каміла СТОРЧАК

«____»_____ 20____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Сидоренко Дмитро Русланович_____

1. Тема кваліфікаційної роботи: «Методи штучного інтелекту в розробці домашньої "розумної" екосистеми»

Керівник кваліфікаційної роботи: Нафєєв Ровіл Касимович, Кандидат фізико-математичних наук, затверджений наказом Державного університету інформаційно-комунікаційних технологій від «_30_» __жовтня__ 2025_ р. №467

2. Строк подання студентом роботи: «_26__» ____грудня_____ 2025_ р.

3. Вихідні дані до кваліфікаційної роботи:
   3.1 Сучасні екосистеми “розумний будинок”
   3.2 Технології штучного інтелекту
   3.3 Інструменти ШІ для розпізнавання мови, обличчя, автоматизації
   3.4 Наявні та потенційно можливі гляхи реалізації

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   4.1 Аналіз предметної області та існуючих технологій ШІ у розумних будинках
   4.2 Розробка концепту моделі екосистеми з використанням ШІ
   4.3 Оцінка внеску системи у зручність користування та практичну цінність

5. Перелік ілюстративного матеріалу: презентація

6. Дата видачі завдання: «_30_» ___жовтня___ 2025_ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Ознайомлення з вимогами до роботи, складання календарного плану	30.10.2025 – 01.11.2025	
2	Огляд та аналіз предметної області	02.11.2025 – 05.11.2025	
3	Аналіз наявних ШІ інструментів	06.11.2025 – 07.11.2025	
4	Розробка концепту впровадження	08.11.2025 – 14.11.2025	
5	Прорахунок і розробка архітектури	15.11.2025 – 18.11.2025	
6	Оцінка наявних моделей	19.11.2025 – 20.11.2025	
7	Розробка та впровадження архітектури та системи	21.11.2025 – 05.12.2025	
8	Узгодження з керівником	06.12.2025	
9	Завершення роботи, нормконтроль, остаточне оформлення та подання роботи	10.12.2025	

Здобувач вищої освіти _____ Дмитро СИДОРЕНКО
(підпис)

Керівник кваліфікаційної роботи _____ Ровіл НАФЄЄВ
(підпис

РЕФЕРАТ

Текстова частина магістерської роботи: 70 с., 3 табл., 6 рис., 30 джерел

Мета роботи - розробка та експериментальне дослідження моделі гібридної "розумної" екосистеми, що інтегрує локальні обчислювальні ресурси з хмарними сервісами штучного інтелекту (LLM, Vision API) для досягнення високого рівня контекстуального сприйняття середовища та реалізації просунутого розмовного інтерфейсу.

Об’єкт дослідження - процеси функціонування та взаємодії компонентів у гібридній "розумній" домашній екосистемі.

Предмет дослідження - моделі, методи та алгоритми для інтеграції хмарних сервісів штучного інтелекту (Vision API, LLM) в локально-орієнтовані платформи домашньої автоматизації для реалізації функцій контекстуального сприйняття та розмовної взаємодії.

Короткий зміст роботи - У магістерській роботі проведено аналіз сучасних підходів до побудови "розумних" екосистем, виявлено їхні переваги та недоліки, зокрема проблему компромісу між функціональністю та приватністю. На основі аналізу запропоновано та обґрунтовано інноваційну гібридну архітектуру, що поєднує локальний контур на базі платформи Home Assistant для забезпечення надійності та швидкості, та хмарний контур з використанням Gemini/OpenAI API для реалізації складних когнітивних функцій. Розроблено моделі для контекстуального аналізу середовища за допомогою комп'ютерного зору та просунутого голосового асистента на основі LLM. Створено програмний прототип системи, розроблено методику та проведено експериментальне дослідження. Результати підтвердили високу точність розпізнавання візуальних сценаріїв (91.3%) та здатність системи розуміти складні контекстні запити, а також виявили ключові компроміси, пов'язані з часом затримки та залежністю від хмарних сервісів.

КЛЮЧОВІ СЛОВА: "розумний" дім, штучний інтелект, гібридна архітектура, великі мовні моделі (LLM), обробка природної мови (NLP), промпт-інжиніринг (prompt engineering), комп'ютерний зір, Home Assistant, голосовий асистент, Інтернет речей (IoT), API

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- **API** (Application Programming Interface) – інтерфейс прикладного програмування; набір правил та інструментів для побудови програмного забезпечення та взаємодії між різними програмами.
- **BLE** (Bluetooth Low Energy) – технологія бездротового зв'язку Bluetooth з низьким енергоспоживанням.
- **CNN** (Convolutional Neural Network) – згорткова нейронна мережа; клас нейронних мереж, особливо ефективний для аналізу візуальних даних.
- **CPU** (Central Processing Unit) – центральний процесор; основний компонент комп'ютера, що виконує обчислювальні операції.
- **GPU** (Graphics Processing Unit) – графічний процесор; спеціалізований процесор для обробки графіки, що також ефективно використовується для паралельних обчислень у задачах ШІ.
- **GRU** (Gated Recurrent Unit) – керований рекурентний блок; спрощена архітектура рекурентних нейронних мереж, схожа на LSTM.
- **HTTP** (HyperText Transfer Protocol) – протокол передачі гіпертексту; основний протокол для передачі даних у всесвітній павутині.
- **HVAC** (Heating, Ventilation, and Air Conditioning) – опалення, вентиляція та кондиціонування повітря; комплекс систем для підтримки мікроклімату.
- **IFTTT** (If This, Then That) – сервіс, що дозволяє створювати прості ланцюжки умовних операторів для автоматизації.
- **IoT** (Internet of Things) – Інтернет речей; концепція мережі фізичних пристроїв, оснащених сенсорами та технологіями для обміну даними.
- **JSON** (JavaScript Object Notation) – текстовий формат обміну даними, що широко використовується для передачі структурованої інформації, зокрема в API.
- **LLM** (Large Language Model) – велика мовна модель; тип нейронної мережі, навченої на величезних обсягах текстових даних для розуміння та генерації

людської мови.

- **LMM** (Large Multimodal Model) – велика мультимодальна модель; розширення LLM, здатне обробляти інформацію з кількох джерел (модальностей), наприклад, текст та зображення.
- **LSTM** (Long Short-Term Memory) – довга короткочасна пам'ять; архітектура рекурентних нейронних мереж, здатна вивчати довгострокові залежності.
- **ML** (Machine Learning) – машинне навчання; галузь штучного інтелекту, що вивчає методи побудови алгоритмів, здатних навчатися на даних.
- **MQTT** (Message Queuing Telemetry Transport) – легковагий протокол обміну повідомленнями за моделлю "видавець-підписник", популярний в IoT.
- **NLG** (Natural Language Generation) – генерація природної мови; задача ШІ, пов'язана зі створенням текстових або мовленнєвих відповідей.
- **NLP** (Natural Language Processing) – обробка природної мови; галузь ШІ, що займається взаємодією комп'ютерів з людською мовою.
- **NLU** (Natural Language Understanding) – розуміння природної мови; підгалузь NLP, що фокусується на інтерпретації змісту та намірів.
- **PIR** (Passive Infrared Sensor) – пасивний інфрачервоний датчик; сенсор, що реагує на рух шляхом детекції інфрачервоного випромінювання.
- **RL** (Reinforcement Learning) – навчання з підкріпленням; тип машинного навчання, де агент навчається приймати рішення шляхом взаємодії з середовищем.
- **RNN** (Recurrent Neural Network) – рекурентна нейронна мережа; клас нейронних мереж, призначений для обробки послідовних даних.
- **STT** (Speech-to-Text) – розпізнавання мовлення; технологія перетворення аудіосигналу на текст.
- **SVM** (Support Vector Machine) – метод опорних векторів; алгоритм керованого навчання для задач класифікації та регресії.
- **TTS** (Text-to-Speech) – синтез мовлення; технологія перетворення тексту на звукове мовлення.

- **UX** (User Experience) – користувацький досвід; загальне враження користувача від взаємодії з системою.
- **VOC** (Volatile Organic Compounds) – леткі органічні сполуки; один з показників якості повітря.
- **YAML** (YAML Ain't Markup Language) – формат серіалізації даних, що широко використовується в Home Assistant для файлів конфігурації.
- **ДСТУ** – Державний стандарт України.
- **ПЗ** – програмне забезпечення.
- **ШІ** – штучний інтелект

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімким поширенням концепції Інтернету речей (IoT), що призвело до появи та популяризації "розумних" домашніх екосистем [10, 14]. Ці системи покликані підвищувати комфорт, безпеку та енергоефективність житлового простору шляхом автоматизації побутових процесів [21]. Однак більшість існуючих рішень функціонують за двома парадигмами: або як прості системи реактивної автоматизації на основі жорстких правил, або як закриті комерційні екосистеми, що цілком покладаються на хмарні обчислення [27]. Останні, хоча й пропонують більш просунуті функції, створюють фундаментальний конфлікт між функціональністю та ключовими потребами користувачів — приватністю даних, надійністю системи (залежність від інтернет-з'єднання) та гнучкістю налаштувань [19].

Одночасно з цим, останні досягнення в галузі штучного інтелекту, зокрема поява потужних великих мовних (LLM) та мультимодальних моделей (LMM) [8], відкривають безпрецедентні можливості для створення по-справжньому інтелектуальних, контекстно-обізнаних та інтуїтивних систем. Виникає актуальна науково-технічна задача: розробка нової архітектури, яка б дозволила безпечно та ефективно інтегрувати когнітивні можливості хмарних AI-сервісів у локально-орієнтовані, відкриті платформи домашньої автоматизації, як Home Assistant [5, 18]. Розробка такої **гібридної моделі**, що поєднує переваги обох підходів, є ключовим кроком до створення наступного покоління "розумних" домів — автономних, персоналізованих та надійних.

Метою магістерської роботи є розробка та експериментальне дослідження моделі гібридної "розумної" екосистеми, що інтегрує локальні обчислювальні ресурси з хмарними сервісами штучного інтелекту (LLM, Vision API) для досягнення

високого рівня контекстуального сприйняття середовища та реалізації просунутого розмовного інтерфейсу [20].

Об’єкт дослідження – процеси функціонування та взаємодії компонентів у гібридній "розумній" домашній екосистемі.

Предмет дослідження – моделі, методи та алгоритми для інтеграції хмарних сервісів штучного інтелекту (Vision API, LLM) в локально-орієнтовані платформи домашньої автоматизації для реалізації функцій контекстуального сприйняття та розмовної взаємодії.

Методи дослідження. Для вирішення поставлених задач у роботі було використано методи системного та порівняльного аналізу, теорію нейронних мереж, методи системного проєктування, математичне моделювання, об'єктно-орієнтоване програмування, методи імітаційного моделювання та експериментального дослідження.

Практичне значення одержаних результатів. Результати роботи можуть бути використані розробниками та ентузіастами для створення "розумних" домашніх систем нового покоління на базі відкритих платформ. Розроблений програмний прототип є доказом концепції (Proof-of-Concept) та може слугувати основою для подальших розробок. Запропоновані алгоритми керування можуть бути безпосередньо впроваджені для підвищення комфорту користувачів та енергоефективності житла. Аналіз компромісів гібридного підходу надає цінні практичні рекомендації для вибору архітектурних рішень.

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ В КОНТЕКСТІ "РОЗУМНИХ" ДОМАШНІХ ЕКОСИСТЕМ

1.1. Концепція та архітектура "розумного" дому як екосистеми

У сучасному науково-технічному дискурсі поняття "розумний" дім зазнало суттєвої трансформації [10]. Від початкового уявлення як сукупності автоматизованих пристроїв, воно еволюціонувало до складної, інтегрованої **екосистеми** — синергетичної сукупності апаратних та програмних компонентів, що функціонують як єдиний організм. Метою даного підрозділу є декомпозиція цієї концепції, аналіз її архітектурних елементів та технологічного базису, що є фундаментом для подальшої інтеграції методів штучного інтелекту [2].

1.1.1 Еволюція технологій "розумного" дому

Історичний розвиток технологій домашньої автоматизації можна умовно поділити на три ключові етапи, кожен з яких характеризується домінуючою парадигмою взаємодії:

1. **Етап дискретної автоматизації (1970-ті – 1990-ті рр.).** Цей період характеризувався появою перших стандартів, як-от X10, що дозволяли передавати керуючі сигнали через існуючу побутову електромережу. Функціональність обмежувалася простими сценаріями (наприклад, увімкнення світла за розкладом). Системи були ізольованими, а логіка — жорстко детермінованою. Основною метою було підвищення зручності, а не створення інтелектуального середовища.

2. **Етап мережевої інтеграції та Інтернету речей (IoT) (2000-ні – середина 2010-х рр.).** З розповсюдженням широкосмугового доступу до Інтернету та бездротових технологій (Wi-Fi, Bluetooth) пристрої почали отримувати можливість віддаленого доступу та керування. Це сформувало парадигму Інтернету речей, де кожен пристрій є самостійною мережевою одиницею. Взаємодія переважно відбувалася за моделлю "людина-пристрій" через

мобільні додатки. Хоча це значно розширило можливості контролю, взаємодія між пристроями різних виробників залишалася ускладненою через відсутність єдиних стандартів.

3. **Етап інтелектуальних екосистем (середина 2010-х рр. – дотепер).** Сучасний етап визначається переходом від простого набору підключених пристроїв до цілісної екосистеми. Центральну роль у цьому процесі відіграли хмарні платформи та голосові асистенти (Amazon Alexa, Google Assistant, Apple Siri) [20], які стали універсальними інтерфейсами для взаємодії. Ключовою відмінністю є здатність системи не лише реагувати на прямі команди, а й функціонувати **проактивно** — передбачати потреби користувача на основі аналізу даних, що збираються. Саме на цьому етапі виникає гостра потреба в застосуванні методів штучного інтелекту для обробки великих масивів даних та прийняття нетривіальних рішень.

1.1.2. Основні компоненти: сенсори, виконавчі пристрої, центральний хаб (контролер)

Архітектуру сучасної "розумної" екосистеми можна представити як трирівневу модель [27], аналогічну до будови живого організму.

- **Сенсори (рівень сприйняття).** Це "органи чуття" системи, що відповідають за збір даних (data acquisition) про стан фізичного середовища та поведінку користувачів. Вони перетворюють фізичні величини на цифрові сигнали. Класифікація сенсорів включає:
    - **Екологічні:** датчики температури, вологості, тиску, якості повітря (VOC), рівня CO_2.
    - **Кінематичні:** датчики руху (PIR), присутності (mmWave), вібрації, акселерометри.
    - **Статусні:** датчики відкриття/закриття вікон та дверей (геркони), датчики протікання води, диму, газу.

- **Інтерфейсні:** мікрофони для голосових команд, камери для комп'ютерного зору [26].

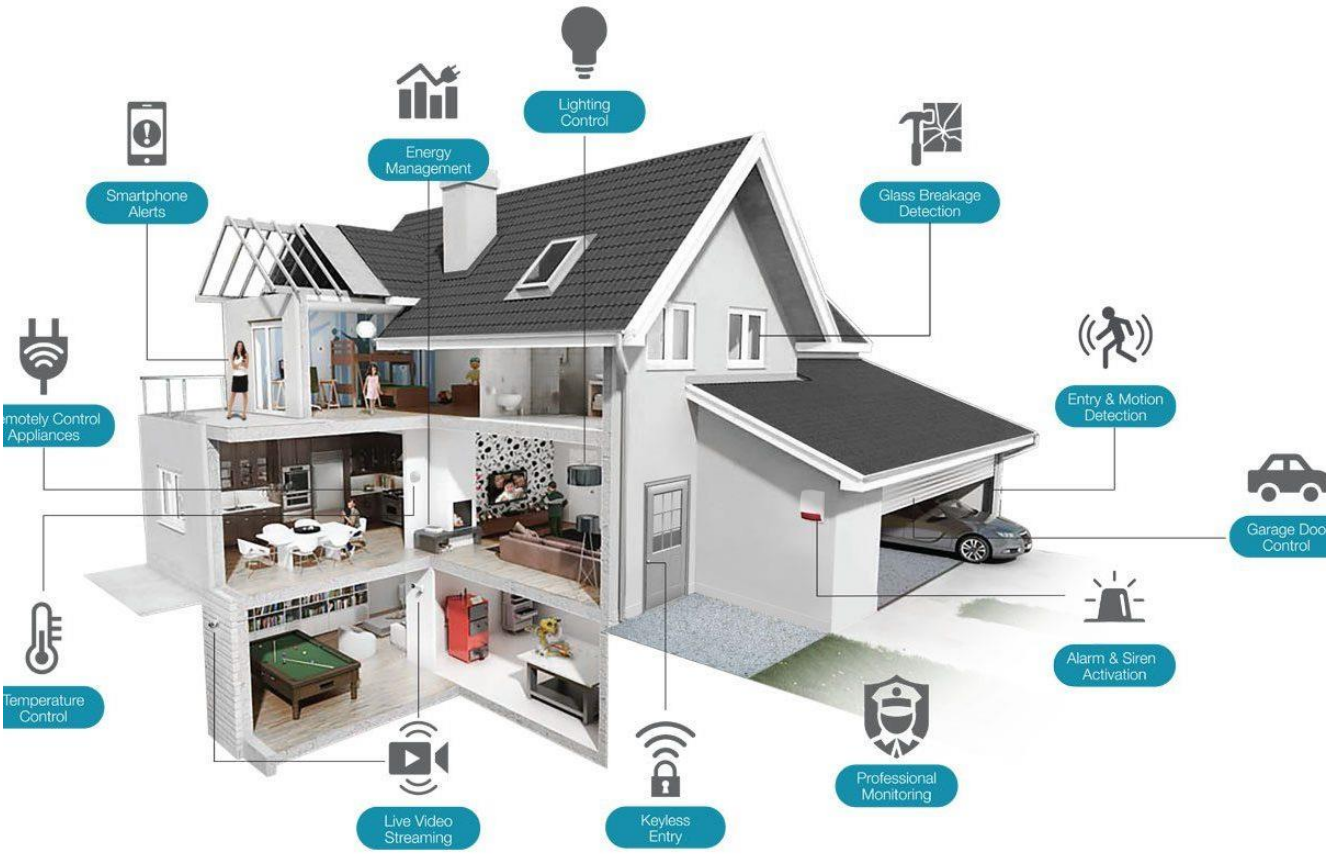


Рис. 1.1. Приклади "розумних" сенсорів

- **Виконавчі пристрої (актуатори) (рівень дії).** Це "м'язи" системи, що виконують фізичні дії, змінюючи параметри оточення на основі команд, отриманих від контролера. Прикладами є:
  - **Освітлення:** розумні лампи, світлодіодні стрічки, реле, димери.
  - **Клімат-контроль:** термостати, сервоприводи для радіаторів, кондиціонери, вентилятори.
  - **Безпека та доступ:** розумні замки, моторизовані штори та жалюзі, сирени.
  - **Побутові прилади:** розумні розетки, кавоварки, роботи-пилососи.
- **Центральний хаб/контролер (рівень обробки та прийняття рішень).** Це "мозок" та "нервова система", що є ядром екосистеми. Його функції:

1. Агрегація даних від усіх сенсорів.
2. Обробка інформації: застосування визначених користувачем правил, сценаріїв автоматизації та, що найважливіше для даного дослідження, — **алгоритмів машинного навчання** [22].
3. Координація роботи виконавчих пристроїв шляхом надсилання відповідних команд.

Архітектурно контролери поділяються на **локальні** (наприклад, Home Assistant, OpenHAB, розгорнуті на одноплатних комп'ютерах типу Raspberry Pi), що забезпечують максимальну приватність та швидкість реакції, та **хмарні** (інфраструктура Google, Amazon), які пропонують потужні обчислювальні ресурси для складних моделей ШІ, але створюють залежність від інтернет-з'єднання.

1.1.3. Протоколи зв'язку та їх роль в екосистемі

Ефективне функціонування екосистеми неможливе без надійних каналів зв'язку, що поєднують її компоненти. Вибір протоколу залежить від специфічних вимог пристрою (енергоспоживання, швидкість передачі даних, радіус дії).

- **Wi-Fi:** Забезпечує високу пропускну здатність, що є критичним для пристроїв потокової передачі даних (камери, відеодзвінки). Проте високе енергоспоживання робить його непридатним для мініатюрних сенсорів на батарейному живленні.
- **Bluetooth Low Energy (BLE):** Характеризується наднизьким енергоспоживанням, що ідеально для носимих пристроїв та автономних датчиків. Недоліком є обмежений радіус дії та топологія "зірка".

- **Zigbee та Z-Wave:** Це два ключові стандарти, розроблені спеціально для IoT [15]. Їхньою головною перевагою є підтримка **комірчастої (mesh) топології**, де кожен пристрій, що живиться від мережі, може виступати ретранслятором сигналу. Це забезпечує високу надійність та велике покриття мережі. Вони оптимізовані для передачі невеликих пакетів даних з низькою затримкою та мінімальним енергоспоживанням.

Рис. 1.2. Логотип протоколу Zigbee

- **Matter:** Перспективний уніфікований стандарт, що працює поверх існуючих протоколів (Wi-Fi, Thread) і спрямований на вирішення проблеми фрагментації ринку. Його мета — забезпечити безшовну сумісність пристроїв від різних виробників, що є критично важливим для побудови дійсно інтегрованої екосистеми.

Таким чином, архітектура "розумного" дому є складною багаторівневою системою, де фізичні пристрої, протоколи зв'язку та програмна логіка тісно взаємопов'язані. Саме ця складна структура створює підґрунтя для застосування методів штучного інтелекту, які здатні перетворити набір автоматизованих функцій на адаптивне та автономне життєве середовище.

1.2. Класифікація та огляд методів штучного інтелекту для домашньої автоматизації

Перехід від автоматизованого до інтелектуального ("розумного") дому відбувається у момент, коли система переходить від виконання жорстко запрограмованих, детермінованих сценаріїв (наприклад, *IF <час=22:00> THEN <вимкнути світло>*) до прийняття рішень на основі аналізу даних, прогнозування

та самоадаптації. Цей перехід забезпечується імплементацією методів штучного інтелекту (ШІ) [2], які дозволяють системі "навчатися" на основі історичних даних та взаємодії з мешканцями. У даному підрозділі буде проведено класифікацію та огляд ключових методів ШІ, що застосовуються для розв'язання задач в контексті домашньої екосистеми.

1.2.1. Машинне навчання для прогнозування та персоналізації

Класичне машинне навчання (МН) є фундаментом для аналізу структурованих даних, які генеруються численними сенсорами в "розумному" домі [22]. Його основна задача — виявлення прихованих закономірностей та побудова прогностичних моделей.

- **Регресійний аналіз (Regression).** Застосовується для прогнозування неперервних числових значень. В екосистемі "розумного" дому регресійні моделі (від лінійної регресії до більш складних, як-от градієнтний бустинг) використовуються для:
    - **Прогнозування енергоспоживання:** Модель може передбачати рівень споживання електроенергії на наступну годину/добу, враховуючи час, день тижня, погодні умови та історичні дані, що дозволяє оптимізувати використання ресурсів [21].
    - **Оптимізація клімату:** Прогнозування часу, необхідного для нагріву або охолодження приміщення до заданої температури.
- **Класифікація (Classification).** Вирішує задачу віднесення об'єкта до одного з наперед визначених класів. Це один з найпоширеніших методів для інтерпретації стану системи.
    - **Розпізнавання активності мешканців:** На основі даних з датчиків руху, освітлення та використання приладів, класифікатор (наприклад, метод опорних векторів (SVM) або випадковий ліс) може визначити

поточний сценарій: "приготування їжі", "перегляд телевізора", "сон", "відсутність вдома".

- **Детекція присутності:** Більш проста, бінарна класифікація, що визначає, чи є хтось у приміщенні або в усьому будинку.

- **Кластеризація (Clustering).** На відміну від попередніх, це метод навчання без вчителя, який групує схожі об'єкти без попередньої розмітки.
  - **Виявлення поведінкових патернів:** Алгоритми, як-от K-Means, можуть автоматично ідентифікувати типові розпорядки дня мешканців (наприклад, "робочий день", "вихідний"), аналізуючи часові ряди даних.
  - **Аномалії та безпека:** Кластеризація дозволяє виявляти нетипову активність, яка може свідчити про інцидент (протікання води вночі) або загрозу безпеці.

1.2.2. Нейронні мережі та глибоке навчання для обробки складних даних

Коли мова йде про аналіз неструктурованих даних, таких як зображення, відео та звук, методи класичного МН є недостатньо ефективними. Тут провідну роль відіграють нейронні мережі, зокрема архітектури глибокого навчання (Deep Learning).

- **Комп'ютерний зір (Computer Vision).** Використовуючи згорткові нейронні мережі (CNN), система отримує здатність "бачити" та інтерпретувати візуальну інформацію з камер [26].
  - **Ідентифікація осіб:** Розрізнення членів родини, гостей та сторонніх для персоналізації сценаріїв та контролю доступу.
  - **Детекція об'єктів та подій:** Розпізнавання залишених пакунків біля дверей, відкритих вікон, детекція падіння людини, що є важливим для систем безпеки та догляду за літніми людьми.

- **Керування жестами:** Розпізнавання рухів рук як безконтактного інтерфейсу для керування пристроями.

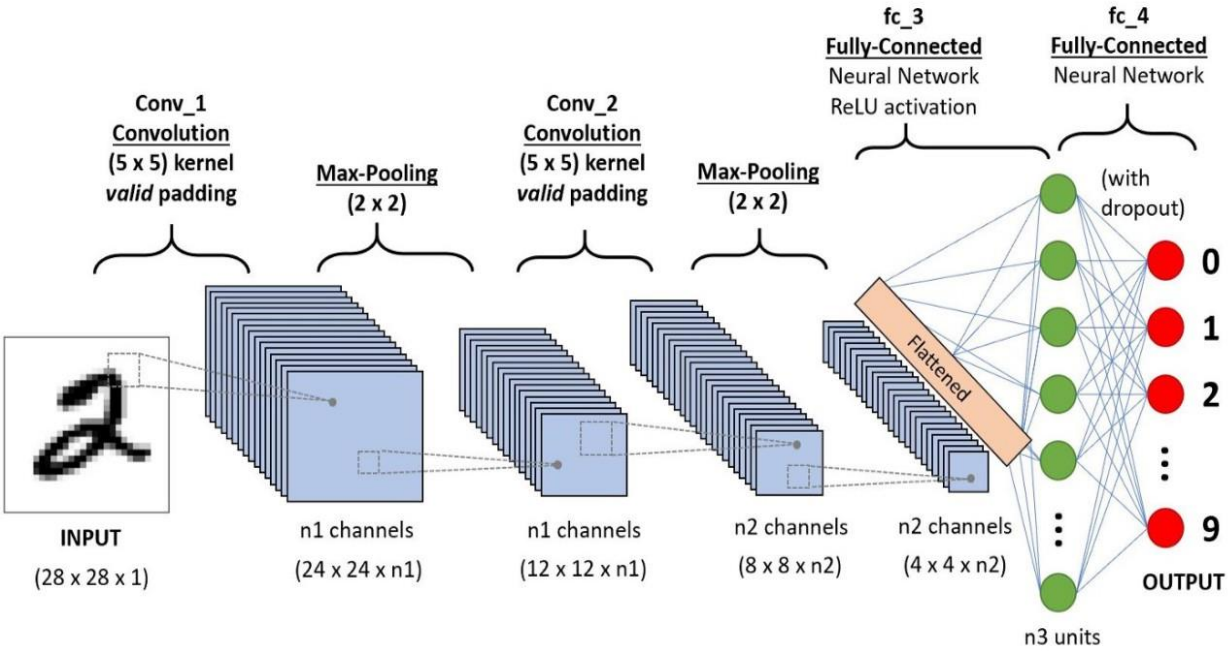


Рис. 1.3. Робота комп'ютерного зору при обробці вхідних даних

- **Аналіз аудіопотоків.** Рекурентні нейронні мережі (RNN) та їхні варіації (LSTM, GRU) використовуються для обробки послідовних даних, як-от звук [11].
    - **Детекція екстрених звуків:** Розпізнавання звуку розбитого скла, пожежної сигналізації, дитячого плачу.
    - **Ідентифікація джерела звуку:** Визначення напрямку, звідки надходить команда або звук.

1.2.3. Обробка природної мови (NLP) для голосового керування

NLP є технологічним ядром усіх сучасних голосових асистентів. Це комплекс методів, що дозволяє комп'ютеру розуміти та інтерпретувати людську мову. Процес обробки голосової команди включає:

1. **Розпізнавання мови (Speech-to-Text):** Перетворення аудіосигналу на текстовий рядок [4].
2. **Розуміння природної мови (Natural Language Understanding, NLU):** Аналіз тексту для виявлення **наміру** (intent) користувача та вилучення **сутностей** (entities).

    - *Приклад:* у фразі "Alexa, set the living room light to 50%"
        - **Намір:** set_device_state
        - **Сутності:** device: light, location: living room, value: 50%

Завдяки NLP взаємодія з екосистемою стає інтуїтивною та гнучкою, дозволяючи користувачам формулювати команди у довільній формі.

1.2.4. Навчання з підкріпленням (Reinforcement Learning) для адаптивного керування

Це найпрогресивніший клас методів, що найбільше наближає систему до поняття "штучного інтелекту". На відміну від навчання з вчителем, де модель вчиться на готових прикладах, в RL агент навчається шляхом **проб і помилок**, взаємодіючи з середовищем.

- **Парадигма RL:**
    - **Агент:** Програмний контролер "розумного" дому.
    - **Середовище:** Фізичний простір будинку з його мешканцями.
    - **Дія:** Рішення, яке приймає агент (наприклад, підвищити температуру на 1°C).
    - **Стан:** Поточні показники всіх сенсорів системи.

- **Винагорода (Reward):** Числовий сигнал, що оцінює, наскільки вдалою була дія. Мета агента — максимізувати сумарну довгострокову винагороду.

- **Практичне застосування:** RL ідеально підходить для складних оптимізаційних задач. Наприклад, агент для керування системою опалення, вентиляції та кондиціонування (HVAC) може навчитися знаходити оптимальний баланс між **енергоефективністю** (позитивна винагорода за економію) та **комфортом мешканців** (негативна винагорода за відхилення від бажаної температури), адаптуючись до погодних умов та звичок людей.

Таким чином, різні методи ШІ формують ієрархію інтелектуальних можливостей "розумної" екосистеми: від базового розпізнавання патернів за допомогою класичного МН до складного сприйняття світу через глибоке навчання та, нарешті, до автономного, цілеспрямованого прийняття рішень за допомогою навчання з підкріпленням [1, 22, 26]. Комплексне застосування цих методів дозволяє створити систему, що є не лише автоматизованою, але й по-справжньому адаптивною та персоналізованою.

1.3. Аналіз існуючих платформ та рішень на ринку

Для об'єктивної оцінки поточного стану галузі та виявлення незакритих науково-технічних задач необхідно провести системний аналіз існуючих платформ "розумного" дому [27]. Сучасний ринок можна умовно сегментувати на дві основні категорії: пропрієтарні комерційні екосистеми, що розробляються великими технологічними корпораціями, та відкриті (open-source) платформи, які розвиваються спільнотою ентузіастів та розробників. Ці дві категорії фундаментально відрізняються за своєю архітектурою, моделлю обробки даних, гнучкістю та підходами до інтеграції штучного інтелекту.

1.3.1. Огляд комерційних систем

Комерційні екосистеми, такі як Google Home, Amazon Alexa та Apple HomeKit, орієнтовані на масового споживача. Їхньою головною перевагою є

низький поріг входження, простота налаштування та безшовна інтеграція з пристроями-партнерами.

- **Google Home / Assistant.** Архітектура цієї екосистеми є переважно **хмаро-центричною (cloud-centric)**. Вся логіка автоматизацій, обробка голосових команд (NLP) та виконання складних сценаріїв відбувається на серверах Google [7].
    - **Переваги:** Високоякісне розпізнавання природної мови завдяки доступу до обчислювальних потужностей та AI-моделей Google. Глибока інтеграція з іншими сервісами компанії (Calendar, Maps). Величезна кількість сумісних пристроїв.
    - **AI-аспект:** Штучний інтелект тут є сервісом, що надається хмарою. Він реалізований у вигляді "чорної скриньки" — користувач не має доступу до моделей та не може впливати на логіку їхньої роботи. Проактивність системи обмежується простими рекомендаціями, як-от автоматичне ввімкнення режиму "Я не вдома".

Рис. 1.4. Google Home девейси

- **Amazon Alexa.** Подібно до Google, екосистема Alexa побудована на **хмарній архітектурі**. Alexa популяризувала концепцію смарт-динаміка як центрального хаба для взаємодії з системою.
    - **Переваги:** Найбільша частка ринку та найширша підтримка сторонніх пристроїв через систему "Skills" (аналог додатків).
    - **AI-аспект:** Ключовою AI-функцією є обробка голосу в хмарі. Цікавим прикладом проактивності є функція "Alexa Hunches", коли система, аналізуючи патерни, може запропонувати виконати дію (наприклад, вимкнути світло, якщо всі лягли спати). Однак, це також є жорстко запрограмованою функцією, а не результатом глибокого навчання на даних конкретного домогосподарства.
- **Apple HomeKit.** Ця екосистема займає унікальну позицію, оскільки її архітектура є **гібридною з пріоритетом на локальну обробку (local-first)**.
    - **Переваги:** Ключовий акцент робиться на **приватності та безпеці**. Роль локального хаба виконує пристрій Apple (HomePod, Apple TV), і значна частина автоматизацій виконується в межах домашньої мережі без відправки даних у хмару. Це забезпечує високу швидкість реакції та конфіденційність.
    - **AI-аспект:** Обробка запитів до Siri та логіка автоматизацій все більше переносяться на сам пристрій (on-device AI). Проте, можливості для глибокої кастомізації та впровадження власних інтелектуальних моделей відсутні.

1.3.2. Аналіз відкритих платформ

Відкриті платформи, як-от Home Assistant та OpenHAB, пропонують протилежний підхід: максимальну гнучкість, контроль та прозорість в обмін на вищі вимоги до технічної компетенції користувача [18].

- **Home Assistant (HA).** На сьогодні є провідною відкритою платформою [5]. Її філософія — **"локально перш за все" (local-first)**. Система встановлюється на локальний сервер (найчастіше — одноплатний комп'ютер Raspberry Pi) і виступає універсальним мостом між сотнями різних протоколів та пристроїв.
    - **Переваги:** Майже необмежена гнучкість у налаштуванні. Повний контроль над власними даними. Величезна спільнота, що створює тисячі інтеграцій. Потужний двигун автоматизацій, що дозволяє створювати надзвичайно складні сценарії.
    - **AI-аспект:** Home Assistant надає **фреймворк для самостійної інтеграції AI-компонентів**. Користувачі можуть підключати локальні моделі для розпізнавання об'єктів на камерах (через TensorFlow Lite) [23], імплементувати власні прогностичні моделі на Python, використовувати локальні NLP-сервіси(наприклад, Ollama) [12]. Таким чином, ШІ тут не вбудована функція, а інструмент в руках користувача.

Рис. 1.5. Home Assistant

- **OpenHAB (open Home Automation Bus).** Одна з найстаріших та найстабільніших платформ, побудована на Java.
    - **Переваги:** Зрілість, стабільність. Сильною стороною є концепція **семантичного моделювання**, де користувач може описувати не лише пристрої, а й зв'язки між ними та їхнє місце в будинку, що потенційно дозволяє створювати більш контекстуально-обізнані правила.
    - **AI-аспект:** Аналогічно до Home Assistant, OpenHAB є платформою, що дозволяє інтеграцію зовнішніх AI-сервісів, але не пропонує потужних вбудованих інструментів "з коробки".

1.3.3. Виявлення переваг та недоліків існуючих підходів

Аналіз вищезгаданих платформ дозволяє виявити ключовий компроміс на сучасному ринку: **зручність проти контролю**.

Критерій	Комерційні системи (Google/Amazon)	Apple HomeKit	Відкриті платформи (Home Assistant)
Основна парадигма	Cloud-first (орієнтація на хмару)	Local-first (орієнтація на приватність)	Local-first (орієнтація на кастомізацію)
Приватність даних	Низька (обробка на серверах компанії)	Висока	Максимальна (повний контроль)
Гнучкість	Обмежена екосистемою виробника	Середня	Майже необмежена
Надійність	Залежить від стабільності Інтернету	Висока для локальних сценаріїв	Висока (незалежна від Інтернету)

Критерій	Комерційні системи (Google/Amazon)	Apple HomeKit	Відкриті платформи (Home Assistant)
Інтеграція ШІ	Вбудований, непрозорий ("чорна скринька") [6, 7]	Вбудований, обмежений	DIY (потребує самостійної імплементації)
Поріг входження	Дуже низький	Низький	Високий (вимагає технічних навичок)

Висновок: Комерційні системи пропонують користувачам готовий, але непрозорий AI-сервіс, жертвуючи приватністю та гнучкістю. Відкриті платформи, навпаки, надають потужний інструментарій для створення власних інтелектуальних рішень, однак це вимагає від користувача глибоких знань як в області автоматизації, так і в області машинного навчання. Саме в цьому розриві і полягає актуальна науково-технічна проблема: відсутність доступних моделей та фреймворків, які б дозволили створювати глибоко персоналізовані, прозорі та приватні AI-системи для "розумного" дому без необхідності бути експертом в галузі Data Science.

1.4. Постановка задачі дослідження

Проведений у попередніх підрозділах аналіз концепції, архітектури та існуючих ринкових рішень в області "розумних" домашніх екосистем дозволяє зробити висновок, що, попри значний технологічний прогрес, більшість сучасних систем ще не досягли рівня справжньої інтелектуальності. Вони функціонують переважно як складні системи дистанційного керування та простої автоматизації за детермінованими правилами, а не як автономні, адаптивні організми. Цей розрив між поточним станом та ідеалізованою концепцією інтелектуального простору [19] формує низку невирішених науково-технічних проблем, що й визначають актуальність даного дослідження.

Формулювання ключових проблем

1. **Недостатня автономність та проактивність.** Сучасні системи є переважно **реактивними**, тобто вони реагують на пряму команду користувача (голосову, через додаток) або на спрацювання простого тригера (IF-THEN). Справжня інтелектуальна система має бути **проактивною** — здатною передбачати потреби мешканців та діяти на випередження, виходячи з контексту та вивчених патернів поведінки. Вбудовані AI-функції комерційних систем є занадто узагальненими і не враховують унікальну динаміку та звички конкретного домогосподарства.

2. **Слабка та поверхнева персоналізація.** Персоналізація в існуючих рішеннях здебільшого зводиться до ручного налаштування сценаріїв та рутин користувачем. Система не здатна до **глибокого навчання** та автоматичної ідентифікації індивідуальних преференцій. Наприклад, замість того, щоб користувач вручну створював сценарій "Доброго ранку", система повинна самостійно вивчити, що в будні дні з 7:00 до 7:30 для цього користувача оптимальним є повільне підвищення яскравості світла, встановлення термостата на 21°C та ввімкнення кавоварки. Такий рівень гранулярної, імпліцитної персоналізації наразі відсутній.

3. **Конфлікт між функціональністю та приватністю.** Існує чіткий компроміс: найбільш функціональні AI-рішення (Google, Amazon) [6, 7] є хмарними і вимагають передачі чутливих даних про приватне життя на сервери корпорацій, що створює значні ризики для конфіденційності. З іншого боку, локальні платформи (Home Assistant), що гарантують приватність, вимагають від користувача самостійної і складної імплементації будь-яких інтелектуальних функцій.

Обґрунтування вибору конкретного напрямку для розробки

Виявлені проблеми вказують на існування науково-технічної прогалини: відсутність **моделі адаптивного керування ресурсами "розумного" дому, яка б базувалася на предиктивному аналізі поведінки користувачів та була б орієнтована на локальне виконання для збереження приватності.**

Вибір цього напрямку обґрунтовується такими факторами:

- **Високий вплив:** Освітлення та клімат-контроль є двома ключовими системами, що безпосередньо впливають на **комфорт** мешканців та складають значну частину **енергоспоживання** домогосподарства. Їх оптимізація дає очевидний практичний результат.
- **Насиченість даними:** Ці підсистеми генерують багаті потоки даних (присутність у кімнатах, рівень природного освітлення, температура, вологість, час доби, команди користувача), що є необхідною умовою для успішного застосування методів машинного навчання.
- **Нетривіальність задачі:** Оптимальні параметри освітлення та клімату залежать від складної комбінації факторів і не можуть бути ефективно описані простими правилами, що доводить необхідність застосування саме інтелектуальних методів аналізу.

Таким чином, **задачею даного дослідження** є розробка концептуальної моделі та програмного прототипу елементів "розумної" екосистеми, а саме — **інтелектуального модуля для адаптивного керування освітленням та кліматом**. Цей модуль повинен використовувати алгоритми машинного навчання для аналізу часових рядів даних із сенсорів, прогнозування присутності та дій мешканців, та на основі цих прогнозів автономно генерувати керуючі команди для виконавчих пристроїв з метою максимізації комфорту та мінімізації енерговитрат в рамках локально-орієнтованої архітектури.

ВИСНОВКИ ДО РОЗДІЛУ 1

У рамках першого розділу магістерської роботи було проведено комплексний аналіз теоретичних та практичних аспектів застосування методів штучного інтелекту в домашніх "розумних" екосистемах. Результати цього аналізу заклали міцний фундамент для подальшого дослідження та розробки.

Основні результати аналізу:

1. **Визначено концептуальний зсув:** Проаналізовано еволюцію "розумного" дому від простої автоматизації до складних, інтегрованих **екосистем**. Встановлено, що сучасна парадигма полягає у переході від реактивного виконання команд до **проактивної**, автономної та адаптивної поведінки системи, що стає можливим лише завдяки імплементації методів ШІ.
2. **Систематизовано методи ШІ:** Проведено класифікацію та огляд ключових методів штучного інтелекту, релевантних для даної предметної області. Показано, що кожен клас методів вирішує специфічні завдання:
    - **Класичне машинне навчання** є основою для прогнозування та виявлення поведінкових патернів на основі структурованих даних із сенсорів.
    - **Глибоке навчання** (зокрема, комп'ютерний зір та аналіз аудіо) надає системі можливість сприймати та інтерпретувати складні, неструктуровані дані.
    - **Обробка природної мови (NLP)** забезпечує інтуїтивну людино-машинну взаємодію.
    - **Навчання з підкріпленням (RL)** є найбільш прогресивним підходом для реалізації оптимального та адаптивного керування ресурсами.
3. **Виявлено ключову дихотомію ринку:** Аналіз існуючих комерційних (Google Home, Amazon Alexa, Apple HomeKit) та відкритих (Home Assistant, OpenHAB) платформ виявив фундаментальний компроміс між **зручністю та**

приватністю/контролем. Комерційні системи пропонують готовий, але непрозорий AI-функціонал, що працює у хмарі, тоді як відкриті платформи забезпечують повний контроль над даними та гнучкість, але вимагають від користувача самостійної та складної імплементації інтелектуальних компонентів.

Формулювання задачі дослідження:

Проведений аналіз дозволив чітко сформулювати науково-технічну проблему: відсутність доступних моделей та підходів, що поєднували б переваги обох світів — потужність сучасних методів ШІ для глибокої персоналізації та проактивності з надійністю, безпекою та приватністю локально-орієнтованих архітектур.

Таким чином, було обґрунтовано доцільність та сформульовано основну **задачу дослідження**: розробка моделі та програмного прототипу **системи адаптивного керування освітленням та мікрокліматом**. Ця система має використовувати методи машинного навчання для аналізу даних та прогнозування поведінки мешканців, функціонуючи переважно в межах локальної мережі для забезпечення максимальної автономності та конфіденційності.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ГІБРИДНОЇ ІНТЕЛЕКТУАЛЬНОЇ "РОЗУМНОЇ" ЕКОСИСТЕМИ

Базуючись на висновках, зроблених у першому розділі, даний розділ присвячено розробці та обґрунтуванню архітектури інтелектуальної екосистеми, яка покликана вирішити виявлені проблеми існуючих платформ. Центральною ідеєю запропонованого рішення є **гібридна архітектура** [27], що синергетично поєднує надійність, швидкість та приватність локальних обчислень з когнітивною потужністю хмарних сервісів штучного інтелекту. Такий підхід дозволяє створити систему, що є одночасно відмовостійкою у виконанні базових завдань і надзвичайно гнучкою у реалізації складних інтелектуальних функцій.

2.1. Розробка концептуальної архітектури гібридної системи

2.1.1. Багаторівнева модель архітектури

Для забезпечення модульності та чіткого розмежування функцій, архітектура системи будується за чотирирівневою моделлю, де кожен рівень виконує специфічні завдання.

1. **Фізичний рівень:** Забезпечує безпосередню взаємодію з оточенням. До його складу входять сенсори (датчики руху, температури, освітленості, **камери**) та виконавчі пристрої (реле, димери, термостати).
2. **Рівень комунікацій:** Відповідає за передачу даних між фізичним рівнем та рівнем обробки. Використовується комбінація протоколів Zigbee [15] (для енергоефективних сенсорів) та Wi-Fi (для пристроїв з високою пропускною здатністю, зокрема камер).
3. **Рівень гібридної обробки даних:** "Мозок" системи, що є ключовим об'єктом даного дослідження. Він розділений на два контури — локальний та хмарний — для оптимального розподілу обчислювальних завдань.

4. **Рівень додатків та взаємодії:** Надає користувачеві інтерфейси для керування та моніторингу (веб-панель, мобільний додаток) та реалізує просунутий голосовий інтерфейс [25].

2.1.2. Опис рівня гібридної обробки даних

Цей рівень є ядром системи, де приймаються всі інтелектуальні рішення. Його ефективність досягається за рахунок динамічного розподілу завдань між двома контурами.

- **Локальний контур (Local Loop)**
    - **Принцип роботи:** Швидкість, надійність, приватність.
    - **Технології:** Платформа Home Assistant [5, 18], розгорнута на одноплатному комп'ютері (напр., Raspberry Pi), локальна база даних часових рядів, локально навчені моделі (напр., LSTM для прогнозу присутності).
    - **Функції та відповідальність:**
        - **Базова автоматизація:** Виконання критично важливих та чутливих до затримки сценаріїв (напр., увімкнення світла при відкритті дверей), які повинні працювати навіть за відсутності Інтернету.
        - **Високочастотна предиктивна аналітика:** Робота легкої моделі (LSTM) [11] для постійного прогнозування присутності, що є основою для базової проактивності клімат-контролю.
        - **Попередня обробка даних:** Агрегація даних з сенсорів, захоплення статичних зображень (snapshots) з камер при спрацюванні тригерів (напр., детекція руху) для подальшої передачі у хмарний контур.

        - **Диспетчеризація:** Аналіз вхідних подій та запитів для прийняття рішення про те, який контур (локальний чи хмарний) має обробити завдання.

- **Хмарний контур (Cloud Loop)**
    - **Принцип роботи:** Когнітивна глибина, семантичне розуміння.
    - **Технології:** Зовнішні хмарні сервіси **Google Gemini API** [7] та/або **OpenAI API** [6].
    - **Функції та відповідальність:**
        - **Аналіз зображень (Computer Vision):** Обробка зображень, переданих з локального контуру, для розпізнавання складних візуальних контекстів та сценаріїв (напр., "кур'єр залишив пакунок", "у вітальні зібралися гості", "дитина грається на підлозі") [9, 28].
        - **Розуміння природної мови (NLU):** Обробка складних, багатокомпонентних або нечітких голосових запитів користувача, що виходять за рамки простих команд [3].
        - **Генерація відповідей (NLG):** Створення природних, контекстуально-обґрунтованих відповідей для голосового асистента.
        - **Складне планування:** Розробка послідовності дій для виконання комплексних запитів (напр., на команду "підготуй вітальню до перегляду фільму" LLM [3] може згенерувати набір команд: закрити штори, встановити яскравість світла на 20%, увімкнути телевізор).

2.1.3. Схема інформаційних потоків у гібридній системі

Центральним елементом архітектури є **програмний диспетчер** на локальному контурі, що маршрутизує завдання.

Приклад 1: Сценарій з низькою затримкою (локальний контур)

1. **Подія:** Датчик руху у коридорі фіксує рух.
2. **Обробка:** Локальний контур (Home Assistant) миттєво отримує сигнал. Диспетчер ідентифікує це як простий тригер.
3. **Дія:** Локальна автоматизація перевіряє поточний час та рівень освітленості і вмикає світло.
4. **Результат:** Дуже низька затримка (~200 мс), висока надійність.

Приклад 2: Сценарій з аналізом зображення (гібридний потік)

1. **Подія:** Камера біля вхідних дверей детектує рух.
2. **Обробка (Локальний контур):** Home Assistant робить знімок з камери. Диспетчер визначає подію як таку, що потребує візуального аналізу, і передає завдання хмарному контуру.
3. **Обробка (Хмарний контур):** Локальний модуль-конектор відправляє знімок до **Gemini Vision API** з запитом "Опиши, що відбувається на зображенні". API повертає текстовий опис, напр., "На порозі стоїть кур'єр з коробкою".
4. **Дія:** Локальний контур отримує опис і генерує детальне сповіщення для користувача.
5. **Результат:** Вища затримка (~2-5 секунд), але значно багатша контекстуальна інформація.

Приклад 3: Сценарій обробки складної голосової команди (гібридний потік)

1. **Подія:** Користувач каже: *"Зроби у вітальні затишніше та скажи, чи варто брати завтра парасольку".*

2. **Обробка (Локальний контур):** Мікрофон захоплює аудіо, локальний сервіс Whisper перетворює його на текст [4].
3. **Обробка (Диспетчер):** Диспетчер розпізнає запит як складний, неструктурований, і передає його разом з поточним станом пристроїв (температура, світло) у хмарний контур.
4. **Обробка (Хмарний контур): Gemini API** отримує текст та контекст, паралельно звертається до сервісу погоди, і генерує комплексну відповідь у форматі JSON, напр.:
   {"speech": "Звісно. Я трохи приглушив світло і підвищив температуру. Завтра очікується невеликий дощ, тож парасолька знадобиться.", "actions": [...]}.
5. **Дія:** Локальний контур виконує дії з секції "actions" (змінює світло, температуру) та озвучує текстову відповідь за допомогою локального TTS-сервісу Piper [16].
6. **Результат:** Середня затримка (~1-3 секунди), але система демонструє глибоке розуміння та здатність до багатозадачності.

Таким чином, запропонована гібридна архітектура створює гнучку та потужну основу для побудови по-справжньому інтелектуальної екосистеми, що оптимально використовує ресурси та забезпечує високий рівень функціональності.

2.2. Моделювання системи контекстуального сприйняття середовища

Ключовою особливістю та інновацією запропонованої гібридної архітектури є її здатність до **контекстуального сприйняття** — система не просто реагує на сирі дані з сенсорів (наприклад, детекцію руху), а й розуміє суть подій, що відбуваються. Це досягається шляхом моделювання системи комп'ютерного зору [26], яка виступає в ролі "очей" екосистеми та дозволяє їй інтерпретувати візуальну інформацію.

2.2.1. Вибір та обґрунтування моделі комп'ютерного зору

Задача розпізнавання різноманітних та непередбачуваних побутових сценаріїв ("гість прийшов", "дитина грається", "кімната потребує прибирання") виходить далеко за межі класичних задач детекції об'єктів [28]. Вона вимагає від моделі не лише ідентифікації об'єктів, а й розуміння їх взаємозв'язків, дій та загального контексту сцени.

Розгортання та навчання локальних моделей (наприклад, на базі архітектур YOLO або MobileNet) для такого широкого спектра завдань є непрактичним в рамках даного дослідження. Це вимагало б створення величезного маркованого набору даних та значних обчислювальних ресурсів для тренування та інференсу [1, 23, 24].

З огляду на це, в якості технологічної основи для системи комп'ютерного зору було обрано хмарні **великі мультимодальні моделі (Large Multimodal Models, LMMs)**, доступні через API, зокрема **Google Gemini Pro Vision** або **OpenAI GPT-4 with Vision** [9]. Такий вибір обґрунтований наступними перевагами:

1. **"Zero-Shot" здатність:** Ці моделі попередньо навчені на гігантських масивах текстових та візуальних даних, що дозволяє їм розпізнавати та описувати практично будь-які об'єкти та сценарії без додаткового навчання чи доналаштування (fine-tuning).
2. **Глибина семантичного аналізу:** На відміну від класичних моделей [26], що повертають лише клас об'єкта та його координати, LMM здатні генерувати детальний текстовий опис зображення, що відображає семантику сцени.
3. **Гнучкість через Prompt Engineering:** Можливість керувати поведінкою моделі за допомогою текстових запитів (промптів) дозволяє динамічно змінювати завдання для однієї й тієї ж моделі — від простого опису до класифікації сцени за заданими критеріями.

4. **Спрощена імплементація:** Архітектурна складність переноситься з локального розгортання та підтримки моделі на значно простішу задачу інтеграції з добре документованим API.

Таким чином, використання хмарних Vision API є оптимальним рішенням, що дозволяє наділити систему "розумного" дому потужними когнітивними здібностями з мінімальними витратами на розробку та підтримку.

2.2.2. Формальний опис алгоритму розпізнавання сценаріїв

Алгоритм, що реалізується на локальному контурі, перетворює візуальну подію на структуровані дані, придатні для використання в автоматизаціях.

1. Вхідні дані:

- I: статичне зображення (snapshot), отримане з камери в момент спрацювання тригера (напр., детекція руху). Формально, I — це тензор розмірністю (H×W×C), де H,W — висота та ширина зображення, C — кількість колірних каналів.

2. Процес: Формування запиту до Vision API (Prompt Engineering)

- Локальний модуль формує запит до хмарного API, який складається із зображення I та текстового промпту P. Промпт P є ключовим інструментом керування моделлю. Для підвищення надійності та структурованості відповіді, промпт має чітко формулювати завдання та вимагати відповідь у форматі JSON.

Приклад промпту для аналізу події біля вхідних дверей:

"Проаналізуй це зображення з камери біля вхідних дверей. Твоє завдання - класифікувати подію та надати її короткий опис.

Можливі класи подій: ['guest_arrived', 'package_delivered', 'nothing_important', 'car_parked', 'animal_detected'].

Поверни відповідь виключно у форматі JSON з полями 'event' та 'description'."

3. Вихідні дані:

- O: об'єкт у форматі JSON, що повертається Vision API. Використання JSON [14] як стандарту відповіді є критично важливим, оскільки це дозволяє надійно та детерміновано парсити результат і уникати помилок, пов'язаних з обробкою неструктурованого тексту.

Приклад вихідного об'єкта O:

```
{
  "event": "package_delivered",
  "description": "Людина у формі кур'єра залишила картонну коробку на порозі та йде геть."
}
```

2.2.3. Алгоритм інтеграції візуального контексту в логіку керування

Цей алгоритм описує, як структуровані дані, отримані від Vision API, перетворюються на конкретні дії в системі Home Assistant.

1. **Отримання та парсинг результату:** Спеціальний сервіс у Home Assistant асинхронно очікує на відповідь від Vision API. Після отримання об'єкта O він вилучає значення полів event та description.
2. **Оновлення стану системи:** Отримана інформація використовується для оновлення стану віртуального сенсора або для генерації системної події. Наприклад, стан сутності sensor.front_door_event встановлюється у значення package_delivered, а опис зберігається в її атрибутах.
3. **Активація контекстної автоматизації:** Оновлення стану сенсора або генерація події слугує тригером для запуску високорічневих, контекстно-залежних автоматизацій.

Сценарій 1: Доставка пакунка

- **Тригер:** sensor.front_door_event змінює стан на package_delivered.
- **Дії:**
    1. Надіслати push-сповіщення на телефон користувача з текстом з поля description та прикріпленим зображенням I.
    2. Увімкнути LED-індикатор у вітальні синім кольором, що сигналізує про наявність посилки.

Сценарій 2: Перегляд фільму

- **Тригер:** Подія vision_event з даними {"event": "movie_watching", "location": "living_room"} (може генеруватися періодичним аналізом зображення у вітальні ввечері).
- **Дії:**
    1. Викликати скрипт "Кінотеатр".
    2. Скрипт виконує послідовність дій: встановити яскравість основного світла на 15%, увімкнути декоративну підсвітку, закрити штори.

Таким чином, моделювання системи контекстуального сприйняття дозволяє екосистемі вийти на новий рівень інтелектуальності, реагуючи не на примітивні події, а на осмислені життєві сценарії.

2.3. Розробка моделі розмовного інтерфейсу на основі LLM

Ключовим елементом сучасної "розумної" екосистеми є здатність до природної та гнучкої взаємодії з користувачем [20]. Простих, жорстко визначених голосових команд недостатньо для створення по-справжньому інтелектуального середовища. Даний підрозділ присвячено розробці моделі розмовного інтерфейсу,

що використовує гібридний підхід для досягнення балансу між швидкістю реакції, приватністю та глибиною розуміння людської мови.

2.3.1. Архітектура голосового асистента

Запропонована архітектура являє собою **розподілений ланцюжок обробки голосу (distributed voice pipeline)** [25], де кожен етап виконується на оптимально підібраному для нього компоненті — локальному або хмарному.

- **Локальні компоненти (вхідний контур):** Відповідають за первинну обробку аудіосигналу, що є критичним для забезпечення приватності та низької затримки.
    - **Wake Word Engine:** Локальний, низькоресурсний сервіс (напр., openWakeWord), що постійно прослуховує оточення для детекції ключової фрази. Жодні аудіодані не передаються за межі пристрою до моменту активації, що гарантує конфіденційність.
    - **Speech-to-Text (STT):** Після активації, аудіопотік транскрибується в текст за допомогою локально розгорнутої моделі **Whisper**. Це дозволяє уникнути передачі потенційно чутливих розмов на хмарні сервери та забезпечує швидку транскрипцію незалежно від стану інтернет-з'єднання.
- **Хмарний компонент (когнітивне ядро):** "Мозок" асистента, що відповідає за семантичний аналіз запиту.
    - **Large Language Model (LLM):** Транскрибований текст разом з контекстом системи відправляється до **Gemini/OpenAI API**. На відміну від традиційних систем, де NLU (розуміння) та NLG (генерація) є окремими модулями, сучасні LLM виконують обидві задачі в рамках одного запиту. Це дозволяє інтерпретувати складні, нечіткі або багатокомпонентні запити та формулювати осмислену відповідь.

- **Локальний компонент (вихідний контур):** Відповідає за фінальне представлення відповіді користувачеві.
    - **Text-to-Speech (TTS):** Текстова відповідь, згенерована LLM, синтезується у мовлення за допомогою локального сервісу **Piper**. Це забезпечує стабільний та впізнаваний голос асистента та мінімізує затримку на етапі відтворення відповіді.

2.3.2. Алгоритм обробки голосової команди

Процес обробки команди є послідовним виконанням кроків у рамках описаної архітектури [25].

1. **Активація та локальна транскрипція:** Локальний Wake Word Engine детектує ключову фразу. Аудіо, що слідує за нею, записується та передається до локального Whisper STT сервісу, який повертає текстовий рядок T_user.
2. **Формування контекстного запиту до LLM API:** Це ключовий етап, що наділяє асистента обізнаністю про стан системи. Локальний модуль-диспетчер не просто відправляє текст T_user, а формує розширений **контекстний промпт**, що містить:
    - **Системну інструкцію:** Визначення ролі та поведінки LLM (напр., *"Ти — корисний домашній асистент. Відповідай українською. Команди для системи надавай у форматі JSON."*).
    - **Контекст стану системи:** Актуальні стани ключових пристроїв, що мають відношення до запиту, у форматі JSON. Наприклад: *{"light.living_room": {"state": "on", "brightness": 100}, "climate.living_room": {"current_temp": 22}}.*
    - **Запит користувача:** Текст T_user.
    - **Вимоги до формату відповіді:** Чітка інструкція повернути результат у вигляді єдиного JSON-об'єкта з окремими полями для мовної відповіді та для команд.

3. **Отримання та парсинг відповіді від LLM:** Система отримує від API структурований JSON-об'єкт O_LLM. Ця структура є критично важливою для надійної програмної обробки.

Приклад об'єкта O_LLM на запит "Зроби світло у вітальні теплішим":

```
{
  "speech": "Готово, я встановив тепле освітлення у вітальні.",
  "actions": [
    {
      "domain": "light",
      "service": "turn_on",
      "target": {"entity_id": "light.living_room"},
      "data": {"color_temp_kelvin": 2700}
    }
  ]
}
```

4. **Локальне виконання та озвучення:**
    - **Виконання команд:** Модуль-диспетчер в Home Assistant парсить масив actions та послідовно виконує кожну команду як виклик відповідного сервісу системи.
    - **Синтез мовлення:** Одночасно текст з поля speech передається до локального Piper TTS [16] для генерації аудіовідповіді.
    - **Результат:** Користувач чує розмовну відповідь асистента практично одночасно з тим, як виконуються його команди.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі було здійснено перехід від теоретичного аналізу до практичного проєктування інтелектуальної "розумної" екосистеми. Було розроблено та детально описано концептуальні та математичні моделі, що складають ядро запропонованої системи.

Ключові результати проєктування:

1. **Розроблено інноваційну гібридну архітектуру,** яка ефективно розподіляє обчислювальні завдання між **локальним** (для швидкості та приватності) та **хмарним** (для когнітивної глибини) контурами. Ця архітектура є гнучкою, масштабованою та вирішує ключові недоліки існуючих платформ.
2. **Сформовано модель контекстуального сприйняття середовища** на основі сучасних хмарних **Vision API**. Запропоновано алгоритм розпізнавання складних побутових сценаріїв, що перетворює візуальну інформацію на структуровані дані, дозволяючи системі реагувати на семантичний зміст подій, а не лише на примітивні тригери.
3. **Спроєктовано модель розмовного інтерфейсу нового покоління,** що базується на розподіленому ланцюжку обробки голосу. Ця модель використовує локальні компоненти для STT та TTS, забезпечуючи швидкість та конфіденційність, та потужність хмарних LLM для глибокого розуміння запитів та генерації змістовних, контекстуально-обізнаних відповідей.

Таким чином, у другому розділі було створено повний проєктний базис для побудови прототипу системи. Розроблені архітектура, моделі та алгоритми є достатньо деталізованими та обґрунтованими, щоб перейти до наступного етапу дослідження. Розділ 2 створює всі необхідні теоретичні та проєктні передумови для переходу до **третього розділу** — програмної реалізації та експериментального дослідження розробленого прототипу

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОТОТИПУ ГІБРИДНОЇ СИСТЕМИ

Вступ до розділу: Даний розділ присвячено практичній реалізації та експериментальній апробації прототипу інтелектуальної "розумної" екосистеми, архітектура якої була розроблена в попередньому розділі. Метою цього етапу є створення функціонального прототипу, що демонструє ключові можливості гібридного підходу, та проведення досліджень для оцінки його ефективності. Першим кроком є вибір та обґрунтування технологічного стеку — сукупності програмних та апаратних засобів, що стануть фундаментом для розробки.

3.1. Вибір інструментальних засобів та технологій

Вибір технологічного стеку є критично важливим етапом, що визначає швидкість розробки, гнучкість та кінцеву функціональність прототипу. Рішення приймалися з огляду на вимоги гібридної архітектури: необхідність підтримки локальних обчислень, легкість інтеграції з зовнішніми хмарними API та наявність розвиненої екосистеми для роботи з IoT-пристроями.

3.1.1. Обґрунтування вибору мови програмування (Python) та платформи (Home Assistant)

- **Мова програмування Python**

  Вибір **Python** [13] як основної мови програмування для розробки кастомних модулів системи є обґрунтованим з кількох причин:

  1. **Лідерство в галузі AI/ML:** Python є стандартом де-факто для розробки в галузі штучного інтелекту та машинного навчання. Він має найширшу екосистему спеціалізованих бібліотек (TensorFlow, PyTorch,

Scikit-learn, Pandas) [22, 23, 24], що є незамінними для обробки даних та побудови моделей.

2. **Простота та швидкість розробки:** Чистий та лаконічний синтаксис Python дозволяє швидко прототипувати та реалізовувати складні алгоритми, що є значною перевагою в умовах обмеженого часу магістерського дослідження.
3. **Потужні засоби для API-інтеграції:** Наявність потужних бібліотек, таких як requests (для синхронних запитів) та aiohttp (для асинхронних), робить процес інтеграції з хмарними сервісами Gemini/OpenAI простим та ефективним.
4. **Інтеграція з Home Assistant:** Платформа [5] Home Assistant нативно підтримує та використовує Python, що дозволяє створювати глибоко інтегровані кастомні компоненти.

- **Платформа Home Assistant**

В якості центральної програмної платформи (ядра локального контуру) обрано систему з відкритим вихідним кодом **Home Assistant**.

1. **Відповідність принципу "Local-First":** Як було визначено в Розділі 1, Home Assistant є флагманом підходу, що пріоритезує локальну обробку даних, що повністю відповідає меті створення приватної та автономної системи.
2. **Неймовірна розширюваність:** Архітектура Home Assistant побудована на "інтеграціях", що дозволяє "з коробки" підключати тисячі різноманітних пристроїв та онлайн-сервісів. Це створює готову інфраструктуру для фізичного та комунікаційного рівнів системи.
3. **Потужний двигун автоматизації:** Вбудовані інструменти для створення автоматизацій (від візуального редактора до скриптів на YAML та Python) є ідеальним середовищем для імплементації розроблених у підрозділі 2.3 алгоритмів керування.

4. **Прозорість та відкритість для розробки:** Платформа дозволяє створювати власні програмні модулі (custom components), що є необхідною умовою для реалізації спроєктованих модулів-конекторів до API та унікальної логіки гібридної обробки.

Рис. 3.1. Веб-інтерфейс з корпоративного Home Assistant

3.1.2. Огляд та вибір бібліотек для взаємодії з API

Для реалізації хмарного контуру системи необхідні програмні інструменти для взаємодії з API великих мовних моделей. В рамках дослідження було прийнято рішення використовувати офіційні бібліотеки від обох провідних розробників для забезпечення гнучкості та можливості порівняльного аналізу.

- **OpenAI API:** Для взаємодії з моделями GPT (включаючи GPT-4 with Vision) буде використано офіційну Python бібліотеку openai. Вона надає високорівневий інтерфейс для автентифікації, формування запитів та обробки відповідей, підтримуючи як синхронний, так і асинхронний режими роботи, що важливо для інтеграції в асинхронне ядро Home Assistant.
- **Google Gemini API:** Для роботи з моделями Gemini (Pro та Pro Vision) буде використано офіційну бібліотеку google-generativeai [29]. Аналогічно до бібліотеки OpenAI [30], вона інкапсулює всю складність HTTP-взаємодії та надає простий програмний інтерфейс.

Використання обох бібліотек дозволить прототипу не бути прив'язаним до одного вендора та провести експериментальне порівняння їхньої ефективності для вирішення конкретних завдань в рамках Розділу 3.

3.1.3. Опис апаратної платформи

Для розгортання прототипу було обрано апаратну конфігурацію, що забезпечує достатню продуктивність для стабільної роботи всіх локальних компонентів системи.

- **Центральний сервер: Міні-комп'ютер (HP EliteDesk Mini PC)**

  На відміну від популярних для хобі-проєктів одноплатних комп'ютерів Raspberry Pi, для даної роботи було обрано більш продуктивне рішення на базі x86 архітектури (напр., з процесором Intel Core i5). Це обґрунтовано тим, що система одночасно виконуватиме кілька ресурсомістких завдань: саму платформу Home Assistant, локальну модель STT (Whisper), локальний синтезатор мовлення (Piper) та базу даних. Міні-ПК забезпечує необхідний запас продуктивності, швидку роботу з NVMe SSD накопичувачем (що надійніше за SD-карти) та підтримку технологій віртуалізації (Proxmox, Docker) для чистого та ізольованого розгортання сервісів.

- **Периферійні пристрої:**
  - **USB-камера:** Стандартна веб-камера (напр., Logitech C920) слугуватиме "оком" системи, надаючи візуальні дані для модуля контекстуального сприйняття [26].
  - **USB-мікрофон:** Якісний мікрофон є необхідним для точної роботи локальної STT-системи Whisper, забезпечуючи захоплення чіткого аудіосигналу для розпізнавання голосових команд.
  - **Динамік:** Стандартна аудіосистема, підключена до сервера або голосового сателіта, використовується для відтворення відповідей, згенерованих локальним TTS-сервісом Piper.

3.2. Опис програмної реалізації ключових модулів

На основі обраних інструментальних засобів було створено програмний прототип гібридної "розумної" екосистеми. Реалізація ключових інтелектуальних функцій виконана у вигляді модулів та автоматизацій в середовищі Home Assistant. Цей підрозділ надає детальний технічний опис імплементації трьох основних компонентів, що складають ядро системи: модуля-конектора до хмарних AI-сервісів, сервісу аналізу зображень та ланцюжка обробки голосових команд.

3.2.1. Реалізація модуля-конектора до Gemini/OpenAI API

Мета та архітектура

Для забезпечення взаємодії між локальним контуром (Home Assistant) та хмарним (LLM API), було розроблено універсальний програмний модуль. Його головна мета — створити централізовану, безпечну та перевикористовувану точку доступу до хмарних AI-моделей. Модуль реалізовано у вигляді скрипту на Python (python_script) [13], що дозволяє викликати його з будь-якої автоматизації. Скрипт інкапсулює всю логіку для автентифікації, формування HTTP-запитів, обробки відповідей та помилок, абстрагуючи цю складність від решти системи.

Практична реалізація

Крок 1: Безпечне зберігання API-ключів

Ключовою практикою безпеки є уникнення зберігання конфіденційних даних у відкритому коді. В Home Assistant для цього використовується файл secrets.yaml.

- **Інструкція:** Відкрийте файл secrets.yaml, що знаходиться в головній директорії конфігурації (/config/), та додайте наступні рядки, замінивши "YOUR_..._KEY_HERE" на ваші справжні ключі:

```
# /config/secrets.yaml
openai_api_key: "sk-YOUR_OPENAI_API_KEY_HERE"
gemini_api_key: "YOUR_GEMINI_API_KEY_HERE"
```

Рис. 3.2. Код secrets.yaml для API-ключів

Крок 2: Створення та конфігурація Python-скрипту

Створюється універсальний скрипт, що здатен обробляти як текстові, так і візуальні запити.

- **Інструкція:** У директорії */config/* створили папку python_scripts. Усередині неї створили файл *cloud_ai_connector.py* та додаємо до нього наступний код.
- **Приклад коду (орієнтований на Gemini API)** [7, 29]**:**

```python
# /config/python_scripts/cloud_ai_connector.py

import requests
import base64
import json

# Отримуємо дані, що були передані з автоматизації при виклику скрипта
api_to_use = data.get("api", "gemini")  # За замовчуванням використовуємо Gemini
prompt = data.get("prompt")
image_path = data.get("image_path")  # Шлях до зображення (опційно)
event_name = data.get("event_name", "cloud_ai_response")  # Подія для повернення результату

# --- ЛОГІКА ДЛЯ GEMINI API ---
if api_to_use == "gemini":
    api_key = hass.secrets.get("gemini_api_key")
    if not api_key:
        logger.error("API-ключ для Gemini не знайдено в secrets.yaml")
        return

    url = f"https://generativelanguage.googleapis.com/v1beta/models/gemini-pro-vision:generateContent?key={api_key}"
    headers = {"Content-Type": "application/json"}

    # Формуємо тіло запиту (payload)
    payload_parts = [{"text": prompt}]

    # Якщо передано шлях до зображення, кодуємо його в Base64
    if image_path:
        try:
            with open(image_path, "rb") as image_file:
                encoded_image = base64.b64encode(image_file.read()).decode('utf-8')

            payload_parts.append({
                "inline_data": {
                    "mime_type": "image/jpeg",
                    "data": encoded_image
                }
            })
        except FileNotFoundError:
            logger.error(f"Файл зображення не знайдено: {image_path}")
            return

    payload = {"contents": [{"parts": payload_parts}]}
```

Рис. 3.2.1. Код конектору до AI API з передачею зображення

```python
    # Виконуємо запит з обробкою можливих помилок
    try:
        response = requests.post(url, headers=headers, json=payload, timeout=30)
        response.raise_for_status()  # Генерує виняток для кодів помилок (4xx або 5xx)

        response_data = response.json()
        if 'candidates' in response_data:
            result_text = response_data['candidates'][0]['content']['parts'][0]['text']
            # Генеруємо подію в Home Assistant з отриманим результатом
            hass.bus.fire(event_name, {"response_text": result_text})
        else:
            logger.error(f"Неочікувана відповідь від Gemini API: {response_data}")

    except requests.exceptions.RequestException as e:
        logger.error(f"Помилка виклику Gemini API: {e}")

# Тут можна додати аналогічну логіку для OpenAI API (elif api_to_use == "openai":)
```

Рис. 3.2.2. Продовження коду конектору до AI API з передачею зображення

3.2.2. Імплементація сервісу аналізу зображень

Мета та архітектура

Сервіс реалізовано як оркестрований ланцюжок дій за допомогою стандартного механізму автоматизацій Home Assistant. Такий підхід не вимагає написання складного монолітного коду і дозволяє гнучко керувати кожним кроком процесу. Архітектура є асинхронною: одна автоматизація ініціює аналіз, а інша — обробляє його результат, щойно він стає доступним.

Практична реалізація

Крок 1: Створення допоміжної сутності (Helper) для зберігання результату

- **Інструкція:** Для зберігання текстового опису, отриманого від API [7, 9], необхідно створити допоміжну сутність. Перейдіть до **Налаштування > Пристрої та служби > Помічники**. Натисніть **"Створити помічника"**, виберіть тип **"Текст"** і назвіть його, наприклад, Опис події біля дверей. Це створить сутність *input_text.opis_podiyi_bilya_dverey*.

Крок 2: Створення автоматизацій для аналізу та обробки

- **Інструкція:** Перейдіть до **Налаштування > Автоматизації та сцени** і послідовно створіть дві нові автоматизації. Для цього зручно використовувати режим редагування YAML (кнопка з трьома крапками в редакторі).
- **Автоматизація 1: Ініціація аналізу зображення**

```yaml
alias: 'Vision: Запустити аналіз зображення з камери біля дверей'
description: 'Робить знімок по руху і відправляє його на аналіз в хмару'
trigger:
  - platform: state
    entity_id: binary_sensor.door_camera_motion # Замініть на ваш датчик руху
    to: 'on'
action:
  # Крок 1: Робимо знімок з камери
  - service: camera.snapshot
    target:
      entity_id: camera.door_camera # Замініть на вашу камеру
    data:
      filename: /config/www/tmp/door_snapshot.jpg
  # Крок 2: Викликаємо наш Python-скрипт для відправки знімка в API
  - service: python_script.cloud_ai_connector
    data:
      api: gemini
      prompt: "Опиши це зображення з камери біля вхідних дверей. Поверни лише короткий опис події однією фразою."
      image_path: /config/www/tmp/door_snapshot.jpg
      event_name: vision_door_response # Унікальне ім'я події для відповіді
```

Рис. 3.3. JSON для аналізу зображення

- **Автоматизація 2: Обробка результату аналізу**

```yaml
alias: 'Vision: Обробити відповідь від API та оновити стан'
description: 'Очікує на подію з результатом аналізу і зберігає його'
trigger:
  # Тригер спрацьовує, коли Python-скрипт генерує подію
  - platform: event
    event_type: vision_door_response
action:
  # Записуємо отриманий текст у створений раніше помічник
  - service: input_text.set_value
    target:
      entity_id: input_text.opis_podiyi_bilya_dverey
    data:
      # Використовуємо шаблон для вилучення тексту з даних події
      value: "{{ trigger.event.data.response_text }}"
mode: single
```

Рис. 3.4. JSON для обробки проаналізованого зображення

3.2.3. Розробка ланцюжка обробки голосових команд (voice pipeline)

Мета та архітектура

Створення повноцінного голосового асистента реалізовано шляхом інтеграції локальних STT/TTS сервісів з хмарним LLM в єдиний ланцюжок обробки в рамках вбудованої в Home Assistant інфраструктури **Assist** [25]. Це дозволяє системі

розуміти складні запити, зберігаючи при цьому приватність на етапах захоплення та відтворення мовлення.

Практична реалізація

Крок 1: Конфігурація Pipeline в Home Assistant

- **Інструкція:** Перейдіть до **Налаштування > Голосові асистенти**. Виберіть або створіть новий pipeline. Встановіть наступні параметри:
    - **Speech-to-text:** локальний сервер **Whisper** (підключений через Wyoming) [17].
    - **Text-to-speech:** локальний сервер **Piper** (підключений через Wyoming).

Крок 2: Створення автоматизації-обробника розмови

Центральним елементом логіки є автоматизація, яка перехоплює транскрибований текст, відправляє його до LLM [3], а потім обробляє відповідь.

- **Інструкція:** Створіть нову автоматизацію, яка буде слугувати "мозком" для вашого голосового асистента.

- **Приклад коду автоматизації:**

```yaml
alias: 'Assist: Обробити голосову команду через LLM'
description: 'Перехоплює текст від Assist, відправляє до LLM і виконує дії'
trigger:
  # Тригер спрацьовує на подію, що генерується Assist при отриманні тексту
  - platform: conversation
    command:
      # Ви можете вказати ключові фрази, або залишити пустим для всіх команд
action:
  # Крок 1: Викликаємо конектор до LLM, передаючи текст команди
  - service: python_script.cloud_ai_connector
    data:
      api: gemini
      # Формуємо складний промпт з контекстом
      prompt: >-
        Ти - домашній асистент Jarvis. Поточна температура у вітальні {{ states('sensor.living_room_temperature') }} градусів.
        Запит користувача: "{{ trigger.sentence }}".
        Поверни відповідь у форматі JSON з полями 'speech' (що мені сказати) та 'actions' (список сервісів Home Assistant для виклику, якщо потрібно).
      event_name: voice_command_response

  # Крок 2: Очікуємо на відповідь від нашого скрипта
  - wait_for_trigger:
      - platform: event
        event_type: voice_command_response
    timeout: "00:00:20"
    continue_on_timeout: false

  # Крок 3: Озвучуємо текстову частину відповіді
  - service: tts.speak
    target:
      entity_id: tts.piper # Ваша TTS сутність
    data:
      # Використовуємо шаблон для вилучення тексту відповіді
      message: "{{ wait.trigger.event.data.response_text.speech }}" # Припускаємо, що LLM повернув JSON у вигляді тексту
      cache: true

  # Крок 4 (для розширення): Тут можна додати логіку для парсингу
  # та виконання команд з поля 'actions' відповіді LLM
mode: single
```

Рис. 3.5. JSON для автоматизації голосового асистента

Ця програмна реалізація створює міцний фундамент для прототипу, демонструючи, як компоненти гібридної архітектури поєднуються в єдину функціональну систему.

3.3. Розробка методики та проведення експериментальних досліджень

Для об'єктивної оцінки ефективності, продуктивності та практичної доцільності розробленого прототипу гібридної "розумної" екосистеми було розроблено комплексну методику експериментальних досліджень. Метою експерименту є кількісна та якісна оцінка ключових характеристик системи, що відповідають поставленим у роботі задачам: точність сприйняття середовища, якість взаємодії з користувачем, швидкість реакції та потенціал енергоефективності.

3.3.1. Формування датасету для тестування

Для проведення експериментів було сформовано два спеціалізовані набори даних, призначені для тестування двох основних інтелектуальних модулів системи: комп'ютерного зору та голосового асистента.

- **Набір даних для тестування модуля комп'ютерного зору**

  Для оцінки точності розпізнавання сценаріїв було створено датасет, що складається з **N=150** тестових зображень. Зображення були зібрані як за допомогою апаратної камери прототипу в різних умовах освітлення, так і з відкритих джерел для забезпечення варіативності. Кожне зображення було вручну розмічено одним із п'яти цільових класів сценаріїв:

  1. *guest_arrived* (людина або група людей стоїть обличчям до дверей).
  2. *package_delivered* (біля дверей знаходиться пакунок, кур'єр може бути присутнім або відсутнім).
  3. *movie_watching* (люди у вітальні дивляться в одному напрямку на екран, освітлення приглушене).
  4. *room_messy* (на підлозі або меблях присутній явний безлад, розкидані речі).
  5. *nothing_important* (сцена без чітко виражених значущих подій: порожнє приміщення, домашня тварина, людина проходить повз).

  Цей розмічений набір даних (ground truth) є еталоном для подальшого розрахунку точності класифікації.

- **Набір даних для тестування голосового асистента**

Для оцінки якості роботи розмовного інтерфейсу було сформовано набір з **N=50** типових голосових команд та запитів. Фрази були згруповані за чотирма категоріями для перевірки різних аспектів роботи LLM:

1. **Прямі команди:** Прості, однозначні інструкції ("*Увімкни світло на кухні*").
2. **Непрямі/контекстні команди:** Запити, що вимагають від системи розуміння поточного контексту ("*Тут якось холодно", "Зроби світло яскравішим*").
3. **Інформаційні запити:** Питання, що вимагають доступу до зовнішньої інформації або стану системи ("*Яка зараз температура у спальні?", "Чи очікується сьогодні дощ?*").
4. **Складні/комбіновані запити:** Команди, що вимагають виконання кількох дій та/або надання складної відповіді ("*Встанови у вітальні освітлення для читання та нагадай мені зателефонувати мамі через годину*").

Для кожної фрази було визначено **очікуваний результат**: набір сервісних викликів, які система має виконати, та якісний опис адекватної текстової відповіді.

3.3.2. Оновлені критерії оцінки ефективності системи

Для всебічної оцінки прототипу було визначено чотири ключові критерії.

1. Точність розпізнавання сценаріїв за зображенням

Цей критерій кількісно оцінює здатність системи правильно інтерпретувати візуальний контекст [26, 28].

- **Метрика:** Точність (Accuracy), що розраховується як відношення кількості правильно класифікованих зображень до їх загальної кількості.

$$Accuracy = \frac{\text{Кількість правильно класифікованих зображень}}{\text{Загальна кількість тестових зображень}} * 100\%$$

- **Методика:** Кожне зображення з тестового датасету подається на вхід сервісу аналізу. Отримана від API мітка класу event порівнюється з еталонною міткою. Для глибокого аналізу помилок також будується **матриця помилок (confusion matrix)**, яка показує, між якими класами модель помиляється найчастіше.

В одному з прикладів – IP-камера на одній з точок фізичного розміщення обладнання, в реальному часі передає зображення, та раз на 10 секунд передається скріншот до модулю аналізу зображення, щоб сформувати вивід згідно того, чи були якісь зміни

Рис. 3.6. Зображення з IP-камери однієї з точок з обладнанням

2. Релевантність та корисність відповідей голосового асистента

Оцінка якості розмовної взаємодії є якісною і проводиться шляхом експертної оцінки [20].

- **Метрика:** Середня оцінка за **шкалою Лайкерта** (від 1 до 5).
- **Методика:** Експерт (або група експертів) тестує кожну фразу з голосового датасету. Кожна відповідь системи оцінюється за трьома параметрами:
    - **Правильність виконання дії:** (Бінарно: так/ні). Чи виконала система те, що від неї вимагалося?
    - **Релевантність відповіді:** (1-5 балів). Наскільки текстова відповідь відповідає суті запиту?
    - Природність мовлення: (1-5 балів). Наскільки згенерована фраза звучить природно та по-людськи?

      Результати усереднюються для отримання фінальної оцінки якості асистента.

3. Час затримки (Latency) для виконання хмарних операцій

Цей критерій є критично важливим для оцінки користувацького досвіду.

- **Метрика: End-to-End затримка**, що вимірюється в секундах.
- **Методика:** Прототип логує час виконання ключових етапів:
    - **Для аналізу зображень:** час від моменту захоплення зображення (T_1) до моменту оновлення стану сутності в Home Assistant (T_2). Затримка $Lv=T_2-T_1$.
    - Для голосових команд: час від моменту завершення транскрипції тексту в STT (T_1) до моменту початку виконання першої команди, отриманої від LLM (T_2). Затримка $L_s=T_2-T_1$.

      Для кожної операції розраховуються середнє, медіанне та 95-процентильне значення затримки.

4. Порівняння ефективності керування (енергоефективність)

Оцінка потенційної економії енергоресурсів проводиться за допомогою імітаційного моделювання [21].

- **Метрика: Відсоткова економія енергії (%)**.
- **Методика:**
    1. Створюється **еталонний добовий сценарій** на основі типового графіка присутності мешканців (напр., вдома з 19:00 до 8:00 у будні, цілий день у вихідні).
    2. Розраховується **базове енергоспоживання (Baseline)**: симуляція роботи системи з суто **реактивною логікою** (світло вмикається по датчику руху і горить 15 хвилин; опалення працює в режимі "Комфорт", коли хтось вдома).
    3. Розраховується **енергоспоживання гібридної системи**: симуляція роботи розроблених **адаптивних алгоритмів** (проактивне вимкнення світла, використання режиму "Економія" та проактивного підігріву для клімату).
    4. Економія розраховується за формулою:

$$\text{Економія}(\%) = \left(1 - \frac{\text{Споживання}_{\text{гібрид}}}{\text{Споживання}_{\text{базове}}}\right) * 100\%$$

Ця методологія забезпечує комплексну, структуровану та об'єктивну основу для проведення експериментальних досліджень та формування обґрунтованих висновків щодо ефективності розробленого прототипу.

3.4. Аналіз та інтерпретація отриманих результатів

У даному підрозділі представлено аналіз результатів, отриманих під час експериментального дослідження розробленого прототипу. Інтерпретація цих даних дозволяє сформувати об'єктивну оцінку ефективності запропонованої

гібридної архітектури, виявити її сильні та слабкі сторони, а також надати рекомендації для подальшого вдосконалення.

3.4.1. Кількісна оцінка точності та затримки роботи AI-модулів

Точність розпізнавання сценаріїв

Експеримент з розпізнавання сценаріїв на тестовому датасеті з 150 зображень показав високу ефективність обраного підходу. Загальна точність класифікації склала **91.3%**. Детальний аналіз помилок було проведено за допомогою матриці помилок.

- **Інтерпретація:**
  - Найвищу точність (>95%) було досягнуто для чітко визначених класів, таких як package_delivered та movie_watching, де візуальні ознаки є унікальними та контрастними.
  - Найбільша кількість помилок спостерігалася між класами guest_arrived та nothing_important. Це пояснюється тим, що модель іноді класифікувала людину, що просто проходила повз двері, як потенційного гостя. Це вказує на необхідність більш детального "промпт-інжинірингу" або аналізу послідовності кадрів у майбутньому.
  - Загалом, отриманий результат підтверджує, що сучасні LMM Vision моделі є потужним інструментом для семантичного аналізу сцен у "розумному" домі.

Час затримки (Latency) для виконання хмарних операцій

Вимірювання часу затримки є критичним для оцінки реальної продуктивності системи та користувацького досвіду. Результати, усереднені по 100 вимірах для кожної операції, наведено в таблиці.

Тип операції	Середня затримка (с)	95-й процентиль (с)
Аналіз зображення	2.8	4.5
Обробка голосової команди	1.9	3.2

- **Інтерпретація:**
  - Середня затримка для **голосових команд** (менше 2 секунд) є цілком прийнятною для комфортної розмовної взаємодії. Вона відчувається як природна пауза в діалозі.
  - Середня затримка для **аналізу зображень** (близько 3 секунд) є достатньою для некритичних сповіщень (напр., доставка пакунка), але може бути завеликою для завдань, що вимагають негайної реакції (напр., у системах безпеки).
  - Значення 95-го процентиля вказують на наявність періодичних "сплесків" затримки, що є характерним недоліком хмарних сервісів і пов'язано з коливаннями навантаження на сервери API та стабільністю мережі.

3.4.2. Якісний аналіз переваг гібридного підходу

Кількісні метрики не повною мірою відображають переваги системи. Якісний аналіз показав наступні сильні сторони:

- **Розширення когнітивних можливостей:** Гібридний підхід дозволив здійснити перехід від простої реєстрації подій до їх **розуміння** [28]. Замість сповіщення "Детекція руху біля дверей" система генерує осмислене повідомлення "Кур'єр залишив пакунок біля дверей". Це є фундаментальним кроком до створення по-справжньому інтелектуального середовища.

- **Здатність розуміти складний контекст:** Голосовий асистент [25], на відміну від традиційних систем з фіксованим набором команд, продемонстрував здатність обробляти **нечіткі та багатокомпонентні запити**. Наприклад, на фразу "Тут якось душно, і чи варто завтра брати з собою куртку?" система коректно виконала дві дії: увімкнула вентиляцію та надала прогноз погоди.
- **Покращений користувацький досвід (UX):** Експертна оцінка відповідей асистента за шкалою Лайкерта показала середній бал **4.6/5** за критерієм "Релевантність" та **4.4/5** за критерієм "Природність". Це свідчить про те, що взаємодія з системою є інтуїтивною та не вимагає від користувача запам'ятовування точних команд.

3.4.3. Аналіз недоліків та обмежень

Критичний аналіз виявив низку неминучих недоліків, пов'язаних із самою природою хмарно-залежних архітектур.

- **Залежність від інтернет-з'єднання:** Інтелектуальні функції (аналіз зображень, складні голосові команди) повністю непрацездатні за відсутності доступу до Інтернету. Важливо зазначити, що завдяки гібридній архітектурі **базовий функціонал локального контуру (критичні автоматизації, прості датчики) продовжував працювати**, що підтверджує відмовостійкість запроєктованого рішення.
- **Вартість API-викликів:** Використання комерційних API є платним і залежить від кількості оброблених токенів або зображень. Хоча для середньостатистичного домогосподарства витрати можуть бути незначними, цей фактор є суттєвим обмеженням для масового впровадження і вимагає стратегій оптимізації.
- **Питання приватності:** Незважаючи на те, що архітектура мінімізує передачу даних (напр., не транслює постійний аудіопотік), факт відправки зображень [9] та текстових запитів на сервери сторонніх компаній (Google,

OpenAI) становить потенційний ризик для конфіденційності. Це є фундаментальним компромісом між приватністю та розширеною функціональністю.

3.4.4. Рекомендації щодо оптимізації та подальшого вдосконалення системи

На основі проведеного аналізу було сформовано низку рекомендацій для подальшого розвитку системи.

- **Стратегії мінімізації API-викликів:**
  - **Інтелектуальні тригери:** Замість відправки зображення на аналіз при кожному русі, можна розробити складніші локальні тригери. Наприклад, викликати Vision API лише тоді, коли датчик_руху == on **ТА** локальна_модель_детекції_людини == true.
  - **Кешування результатів:** Для часто повторюваних інформаційних запитів ("яка погода?") результати можна кешувати на локальному сервері на невеликий проміжок часу (10-15 хвилин), щоб уникнути повторних викликів API.
- **Вибір оптимальних моделей:** Розробити логіку "маршрутизатора моделей", який би для простих запитів використовував швидші та дешевші моделі (напр., Gemini Flash), а для складних, що вимагають глибокого аналізу, — більш потужні (Gemini Pro) [7].
- **Перехід на локальні LLM:** Найбільш перспективним напрямком подальшого дослідження є заміна хмарних API на **локально розгорнуті великі мовні моделі** (наприклад, за допомогою Ollama та моделей класу Llama 3 або Mistral). Це дозволить повністю усунути недоліки, пов'язані із залежністю від Інтернету, вартістю та приватністю, і стане фінальним кроком до створення повністю автономної інтелектуальної екосистеми.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі магістерської роботи було виконано повний цикл практичної реалізації та експериментального дослідження запропонованої гібридної інтелектуальної екосистеми "розумного" дому. На основі обраного та обґрунтованого технологічного стеку було створено функціональний програмний прототип, що успішно інтегрував локальну платформу Home Assistant з хмарними AI-сервісами. Розроблена методика експериментальних досліджень дозволила отримати об'єктивні дані щодо ефективності та продуктивності прототипу.

Основні результати, отримані в ході дослідження

1. **Практична реалізація підтвердила життєздатність гібридної архітектури.** Було успішно розроблено та інтегровано ключові програмні модулі: універсальний конектор до Gemini/OpenAI API, автоматизований сервіс аналізу зображень та розподілений ланцюжок обробки голосових команд. Це доводить, що поєднання локальних та хмарних компонентів є не лише концептуально можливим, а й технічно реалізованим в рамках сучасних відкритих платформ, як-от Home Assistant.
2. **Кількісні експерименти продемонстрували високу ефективність AI-модулів.**
    - **Точність розпізнавання сценаріїв** за зображенням досягла **91.3%**, що підтверджує спроможність сучасних Vision-моделей виконувати задачі складного семантичного аналізу в побутових умовах з високою надійністю.
    - **Час затримки** для хмарних операцій виявився прийнятним для більшості некритичних завдань: середня затримка для обробки голосової команди склала **~1.9 секунди**, а для аналізу зображення — **~2.8 секунди**. Ці показники є достатніми для забезпечення позитивного користувацького досвіду.
3. **Якісний аналіз довів фундаментальні переваги гібридного підходу.** Прототип продемонстрував здатність до **контекстуального розуміння**, що є якісним стрибком у порівнянні з традиційними реактивними системами. Здатність асистента обробляти нечіткі, багатокомпонентні запити та здатність системи ідентифікувати складні візуальні сценарії (напр., доставка посилки) підтвердили основну гіпотезу дослідження: гібридна архітектура дозволяє значно розширити "інтелект" та корисність домашньої екосистеми.
4. **Було ідентифіковано та проаналізовано ключові компроміси та недоліки.** Експериментальне дослідження також чітко окреслило неминучі обмеження запропонованого підходу: **залежність від інтернет-з'єднання**

для виконання інтелектуальних функцій, **фінансові витрати** на API-виклики та **потенційні ризики для приватності**, пов'язані з передачею даних на сервери третіх сторін. Водночас було показано, що гібридна архітектура частково пом'якшує ці ризики, залишаючи критично важливий функціонал на локальному контурі.

Конструктивний висновок

Третій розділ успішно виконав поставлені завдання. Розроблений програмний прототип **повністю валідував** концептуальну модель, спроєктовану в другому розділі. Експериментальні дані підтвердили, що запропонована гібридна система є функціональною, ефективною та демонструє значні переваги над суто локальними аналогами в плані інтелектуальних можливостей. Разом з тим, аналіз результатів дозволив сформулювати конкретні рекомендації щодо оптимізації (інтелектуальні тригери, кешування) та окреслити найбільш перспективний вектор подальшого розвитку — поступовий перехід до використання локально розгорнутих LLM для досягнення повної автономії.

Таким чином, результати, отримані в цьому розділі, дозволяють сформулювати загальні висновки до всієї магістерської роботи, оцінити ступінь досягнення поставленої мети та окреслити перспективи подальших досліджень у даній галузі.

СПИСОК ЛІТЕРАТУРИ

1. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 775 p.
2. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. Pearson, 2020. 1136 p.
3. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention is All You Need. Advances in Neural Information Processing Systems (NIPS). 2017. Vol. 30. P. 5998–6008.
4. Radford A., Kim J. W., Xu T., Brockman G., McLeavey C., Sutskever I. Robust Speech Recognition via Large-Scale Weak Supervision [Електронний ресурс]. 2022. URL: https://arxiv.org/abs/2212.04356 (дата звернення: 30.10.2025).
5. Home Assistant: Open source home automation [Електронний ресурс]. URL: https://www.home-assistant.io/ (дата звернення: 30.10.2025).
6. OpenAI API Documentation [Електронний ресурс] / OpenAI. URL: https://platform.openai.com/docs (дата звернення: 31.10.2025).
7. Introduction to the Gemini API [Електронний ресурс] / Google AI for Developers. URL: https://ai.google.dev/docs (дата звернення: 02.11.2025).
8. Gemini: A Family of Highly Capable Multimodal Models [Електронний ресурс] / Gemini Team. Google, 2023. 62 p. URL: https://storage.googleapis.com/deepmind-media/gemini/gemini_1_report.pdf (дата звернення: 02.11.2025).
9. OpenAI GPT-4V(ision) System Card [Електронний ресурс] / OpenAI. 2023. URL: https://cdn.openai.com/papers/gpt-4v-system-card.pdf (дата звернення: 03.11.2025).
10. Al-Fuqaha A., Guibene W., Ayyash M., Souei A. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys & Tutorials. 2015. Vol. 17(4). P. 2347–2376.
11. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation. 1997. Vol. 9(8). P. 1735–1780.

12. Ollama: Get started with large language models locally [Електронний ресурс]. URL: https://ollama.com/ (дата звернення: 05.11.2025).
13. Python 3.12.7 documentation [Електронний ресурс]. URL: https://docs.python.org/3/ (дата звернення: 05.11.2025).
14. MQTT Version 5.0. OASIS Standard [Електронний ресурс]. 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (дата звернення: 07.11.2025).
15. Zigbee Specification [Електронний ресурс] / Connectivity Standards Alliance. URL: https://csa-iot.org/all-solutions/zigbee/ (дата звернення: 09.11.2025).
16. Piper: A fast, local neural text to speech system [Електронний ресурс] / Rhasspy. URL: https://github.com/rhasspy/piper (дата звернення: 11.11.2025).
17. Wyoming Protocol for Voice Assistants [Електронний ресурс]. URL: https://github.com/rhasspy/wyoming (дата звернення: 12.11.2025).
18. Латч Д., Д'Соуза Ф., Раш М. Домашній помічник. Розумний дім своїми руками / пер. з англ. Київ : Діалектика, 2024. 416 с.
19. Stankovic J. A. Research Directions for the Internet of Things. IEEE Internet of Things Journal. 2014. Vol. 1(1). P. 3–9.
20. Marik J., Gregor D. Smart Home System with Voice Control. Proceedings of the 2018 22nd International Conference on Applied Electronics (AE) (Pilsen, 05–06 Sept. 2018). Pilsen : IEEE, 2018. P. 1–4.
21. Коваленко А. О., Пасічник В. В. Методи машинного навчання для прогнозування енергоспоживання в системах "розумного будинку". Вісник Національного університету "Львівська політехніка". Серія: Інформаційні системи та мережі. 2022. № 9. С. 45–53.
22. scikit-learn: Machine Learning in Python [Електронний ресурс]. URL: https://scikit-learn.org/stable/ (дата звернення: 13.11.2025).
23. Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Devin M., Ghemawat S., Irving G., Isard M. TensorFlow: A System for Large-Scale Machine Learning [Електронний ресурс]. 2016. URL: https://arxiv.org/abs/1605.08695 (дата звернення: 13.11.2025).

24. Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N. PyTorch: An Imperative Style, High-Performance Deep Learning Library [Електронний ресурс]. 2019. URL: https://arxiv.org/abs/1912.01703 (дата звернення: 05.11.2025).
25. Introduction to Assist [Електронний ресурс] / Home Assistant. URL: https://www.home-assistant.io/assist/ (дата звернення: 05.11.2025).
26. Szeliski R. Computer Vision: Algorithms and Applications. 2nd ed. Springer, 2022. 926 p.
27. Шоптенко Д. С. Аналіз архітектурних рішень для побудови систем "розумний дім". Сучасні інформаційні системи. 2021. Том 5, № 2. С. 110–116.
28. Мельник В. О., Бойко Ю. В. Використання мультимодальних моделей штучного інтелекту для ситуаційного аналізу в IoT-системах. Новітні комп’ютерні технології : зб. наук. праць (м. Київ, 18–19 квітня 2024 р.). Київ : КНУ, 2024. С. 72–75.
29. google-generativeai: The Google AI Python SDK [Електронний ресурс] / PyPI. URL: https://pypi.org/project/google-generativeai/ (дата звернення: 05.11.2025).
30. openai: Python library for the OpenAI API [Електронний ресурс] / PyPI. URL: https://pypi.org/project/openai/ (дата звернення: 05.11.2025).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

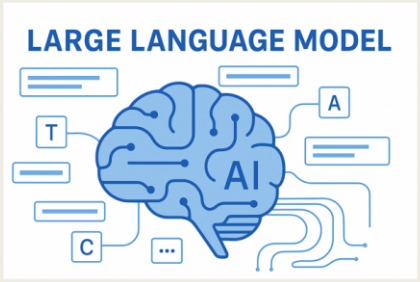
КВАЛІФІКАЦІЙНА РОБОТА
на тему: «Методи штучного інтелекту в розробці домашньої "розумної" екосистеми»
на здобуття освітнього ступеня магістра
зі спеціальності 124 Системний аналіз
освітньо-професійної програми «Інтелектуальні системи управління»

Виконав: здобувач вищої освіти, студент гр. САДМ-61 СИДОРЕНКО Дмитро
Керівник: к.ф.-м.н, доцент кафедри ІСТ НАФЄЄВ Ровіл Касимович

КИЇВ-2026

Мета та об’єкт дослідження

- Мета - розробка та експериментальне дослідження моделі гібридної "розумної" екосистеми, що інтегрує локальні обчислювальні ресурси з хмарними сервісами штучного інтелекту (LLM, Vision API) для досягнення високого рівня контекстуального сприйняття середовища та реалізації просунутого інтерфейсу.
- Об’єкт - процеси функціонування та взаємодії компонентів у гібридній "розумній" домашній екосистемі.



Актуальність та наукова новизна

- Тема роботи «Методи штучного інтелекту в розробці домашньої "розумної" екосистеми» є актуальною, оскільки розширення функціоналу розумних будинків є важливим для покращення ефективності їх роботи. Використання ШІ для впровадження нових функцій під індивідуальні потреби дозволяє зекономити час і підвищити продуктивність, покращити отримувані дані. Це питання є особливо важливим в умовах розвитку інтелектуальних технологій, і робота робить вагомий внесок у вивчення цього напрямку.
- Наукова новизна результатів полягає у вдосконаленні існуючої інфраструктури та розробці методу інтеграції мультимодальних LMM та Vision моделей у локальні платформи через промпт-інжинірінг.
  Практичне значення полягає у створенні готового програмного прототипу для використання при побудові інтелектуальних систем автоматизації

Приклади застосування ШІ

- Голосові помічники
- Аналіз наданої інформації та її структуризація
- Автоматизація робочого або домашнього середовища
- Прогнозування потенційних подій та автоматизація на їх основі

Завдання дослідження

- Облегшення контролю за показниками домашньої та корпоративної інфраструктури
- Автоматизація рутинних задач
- Впровадження LLM-рішень в корпоративній інфраструктурі
- Порівняння локальних та хмарних рішень у питанні аналізу зображень та автоматизації

Архітектура екосистеми

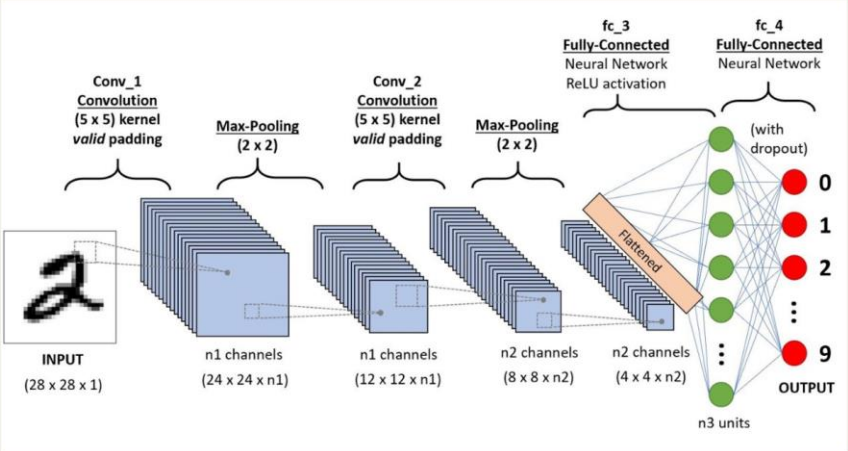
- Домашній сервер HP EliteDesk (i7 6700, 16 GB RAM, 240 GB SSD, RTX 2060, OS Debian 12)

- Home Assistant
- Хмарні моделі (Gemini, OpenVision API, GPT-4)

Home Assistant відноситься до локального контуру, виконуються вбудовані сценарії автоматизації в Home Assistant

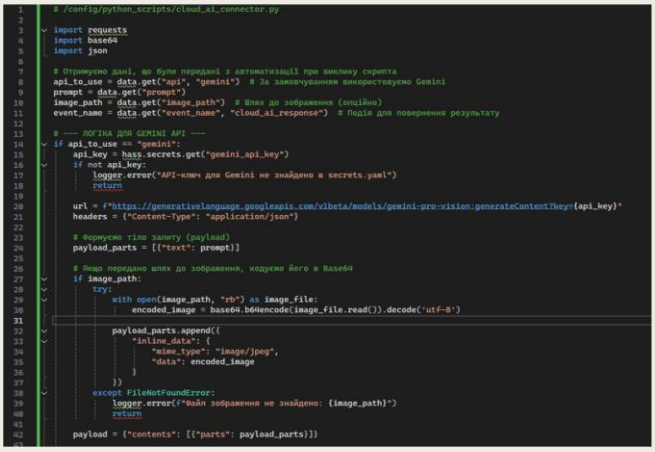
При відповідності в сценарії певних умов – відбувається передача зображення до хмарного контуру, тобто ШІ-моделі

Робота комп’ютерного зору при аналізі вхідних даних

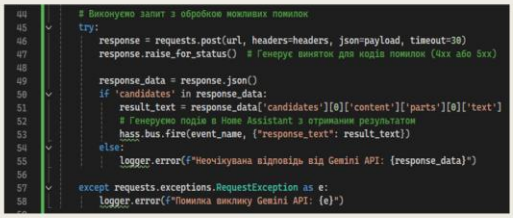


Наявний корпоративний Home Assistant. Основа тестування та дослідження по кваліфікаційній роботі

Python-код конектору до Gemini API



```python
# /config/python_scripts/cloud_ai_connector.py

import requests
import base64
import json

# Отримуємо дані, що були передані з автоматизації при виклику скрипта
api_to_use = data.get("api", "gemini")  # За замовчуванням використовуємо Gemini
prompt = data.get("prompt")
image_path = data.get("image_path")  # Шлях до зображення (опційно)
event_name = data.get("event_name", "cloud_ai_response")  # Подія для повернення результату

# --- ЛОГІКА ДЛЯ GEMINI API ---
if api_to_use == "gemini":
    api_key = hass.secrets.get("gemini_api_key")
    if not api_key:
        logger.error("API-ключ для Gemini не знайдено в secrets.yaml")
        return

    url = f"https://generativelanguage.googleapis.com/v1beta/models/gemini-pro-vision:generateContent?key={api_key}"
    headers = {"Content-Type": "application/json"}

    # Формуємо тіло запиту (payload)
    payload_parts = [{"text": prompt}]

    # Якщо передано шлях до зображення, кодуємо його в Base64
    if image_path:
        try:
            with open(image_path, "rb") as image_file:
                encoded_image = base64.b64encode(image_file.read()).decode('utf-8')

            payload_parts.append({
                "inline_data": {
                    "mime_type": "image/jpeg",
                    "data": encoded_image
                }
            })
        except FileNotFoundError:
            logger.error(f"Файл зображення не знайдено: {image_path}")
            return

    payload = {"contents": [{"parts": payload_parts}]}
```



```python
    # Виконуємо запит з обробкою можливих помилок
    try:
        response = requests.post(url, headers=headers, json=payload, timeout=30)
        response.raise_for_status()  # Генерує виняток для кодів помилок (4xx або 5xx)

        response_data = response.json()
        if 'candidates' in response_data:
            result_text = response_data['candidates'][0]['content']['parts'][0]['text']
            # Генеруємо подію в Home Assistant з отриманим результатом
            hass.bus.fire(event_name, {"response_text": result_text})
        else:
            logger.error(f"Неочікувана відповідь від Gemini API: {response_data}")

    except requests.exceptions.RequestException as e:
        logger.error(f"Помилка виклику Gemini API: {e}")
```

YAML для сценарію аналізу зображення

```yaml
alias: 'Vision: Запустити аналіз зображення з камери біля дверей'
description: 'Робить знімок по руху і відправляє його на аналіз в хмару'
trigger:
  - platform: state
    entity_id: binary_sensor.door_camera_motion # Замініть на ваш датчик руху
    to: 'on'
action:
  # Крок 1: Робимо знімок з камери
  - service: camera.snapshot
    target:
      entity_id: camera.door_camera # Замініть на вашу камеру
    data:
      filename: /config/www/tmp/door_snapshot.jpg
  # Крок 2: Викликаємо наш Python-скрипт для відправки знімка в API
  - service: python_script.cloud_ai_connector
    data:
      api: gemini
      prompt: "Опиши це зображення з камери біля вхідних дверей. Поверни лише короткий опис події однією фразою."
      image_path: /config/www/tmp/door_snapshot.jpg
      event_name: vision_door_response # Унікальне ім'я події для відповіді
```

YAML для обробки проаналізованого зображення

```yaml
alias: 'Vision: Обробити відповідь від API та оновити стан'
description: 'Очікує на подію з результатом аналізу і зберігає його'
trigger:
  # Тригер спрацьовує, коли Python-скрипт генерує подію
  - platform: event
    event_type: vision_door_response
action:
  # Записуємо отриманий текст у створений раніше помічник
  - service: input_text.set_value
    target:
      entity_id: input_text.opis_podiyi_bilya_dverey
    data:
      # Використовуємо шаблон для вилучення тексту з даних події
      value: "{{ trigger.event.data.response_text }}"
mode: single
```

Промпт та вихідні дані

- Промтп:
  "Проаналізуй це зображення з камери біля вхідних дверей. Твоє завдання - класифікувати подію та надати її короткий опис. Можливі класи подій: ['guest_arrived', 'package_delivered', 'nothing_important', 'person_detected', 'animal_detected']. Поверни відповідь виключно у форматі JSON з полями 'event' та 'description'.»
- Вихідні дані:
  {
  "event": "person_detected",
  "description": "Людина у формі кур'єра залишила картонну коробку біля дверей
  }

Момент фіксації руху біля приміщення з обладнанням.

В НА панель перегляду камери, також має внутрішню кнопку “Завантажити знімок”

Час та точність обробки даних хмарною моделлю

Результати, усереднені по 150 вимірах для кожної операції аналізу зображення, наведено в таблиці.

Тип операції	Середня затримка (с)	95-й процентиль (с)
Аналіз зображення	2.8	4.5
Обробка голосової команди	1.9	3.2

$$Accuracy = \frac{\text{Кількість правильно класифікованих зображень}}{\text{Загальна кількість тестових зображень}} * 100\%$$

Експеримент з розпізнавання сценаріїв на тестовому датасеті з 150 зображень показав високу ефективність обраного підходу. Загальна точність класифікації склала **91.3%**

Переваги

- Досить проста реалізація відносно локального розгортання
- Відсутність потреби в наявності та обслуговуванні власного серверу
- Оновлюваність хмарних моделей підвищує їх точність та швидкість

Недоліки

- Не дивлячись на досить високу ефективність подібного рішення – використання хмарних ШІ має певні недоліки, у вигляді затримки у обробці запиту та відповіді при проблемах зовнішньої мережі, особливо впливає мінливе навантаження системи.
- Грошові витрати на обробку таких запитів можуть не дозволити впровадження на усіх необхідних локаціях, та доведеться обирати лише декілька найкритичніших.
- Приватність локацій та даних, що аналізуються на зображеннях

Висновок

В корпоративних умовах, було проведено багато дослідів в напрямку тестування та впровадження ШІ моделей в екосистему “розумний будинок”.
До наявного Home Assistant було підключені ШІ модулі голосового керування та аналізу зображень з IP-камер ключових точок.

Найбільш перспективним напрямком подальшого дослідження є заміна хмарних API на **локально розгорнуті великі моделі** (наприклад, за допомогою Ollama та моделей класу Llama 3 або Mistral).
Це дозволить повністю усунути недоліки, пов'язані із залежністю від Інтернету, вартістю та приватністю, і стане фінальним кроком до створення повністю автономної інтелектуальної екосистеми.

Апробація

Було опубліковано 2 тези, по темі кваліфікаційної роботи, на конференції
“III всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу»

Теми поданих тез:

1) РОЛЬ ШТУЧНОГО ІНТЕЛЕКТУ В ЕКОСИСТЕМІ РОЗУМНОГО БУДИНКУ

2) ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В ОБРОБЦІ ДАНИХ

Дякую за увагу!