ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавр

на тему: **«Застосування фреймворків для синтезу AI-агента»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні
системи та технології
ступеня вищої освіти бакалавр
групи 126ІСТ_бд_31[1]
Вовнянко Іван Владиславович
Керівник: Слюсарь Ігор Іванович
Рецензент: Муравльов Володимир
В’ячеславович

Полтава – 2026 року

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

Освітня програма Інформаційні управляючі системи
Спеціальність 126 Інформаційні системи та технології
Рівень вищої освіти перший (бакалаврський)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Юрій УТКІН
«10» червня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Вовнянка Івана Владиславовича

1. Тема кваліфікаційної роботи:
«Застосування фреймворків для синтезу AI-агента»,
Керівник роботи: к. т. н., доцент, доцент кафедри інформаційних систем та технологій Слюсарь Ігор Іванович.
Затверджено наказом закладу вищої освіти від «10» квітня 2026 року № 384-ст

2. Строк подання здобувачем вищої освіти роботи «08» червня 2026 року.

3. Вихідні дані до роботи: науково-технічна література, аналітичні джерела мережі Інтернет, що містять інформацію про AI-агентів, agentic AI, фроеймворки AI-агентів, LM Studio, n8n, LangChain, AutoGen, CrewAI, LlamaIndex, PicoFlow, PocketFlow, PicoClaw, Docker

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Розділ 1 Особливості застосування фреймворків AI-агентів
Розділ 2. Використання фреймворків для побудови AI-агента
Розділ 3. Рекомендації щодо практичної реалізації AI-агента

5. Перелік графічного матеріалу: схеми, рисунки, діаграми за темою та об’єктом дослідження

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання отримав
Оцінювання економічної ефективності результатів дослідження	Загребельна І. Л., к. е. н., доцент, доцент кафедри економіки та публічного управління	01.04.2026 р.	29.05.2026 р.

7. Дата видачі завдання «10» червня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Вибір і затвердження теми роботи	02.06.2025 р. – 04.06.2025	
2	Складання і затвердження розгорнутого плану та завдання на кваліфікаційну роботу	05.06.2025 р. – 10.06.2025 р.	
3	Опрацювання джерел інформації	11.06.2025 р. – 15.06.2025 р.	
4	Збір, вивчення і обробка інформації, необхідної для виконання роботи	16.06.2025 р. – 20.06.2025 р.	
5	Виконання теоретико-методологічного розділу роботи	08.09.2025 р. – 01.11.2025 р.	
6	Виконання дослідницько-аналітичного розділу роботи	19.01.2026 р. – 14.02.2026 р.	
7	Виконання проєктно-рекомендаційного розділу роботи	16.02.2026 р. – 31.03.2026 р.	
8	Оцінювання економічної ефективності результатів дослідження	01.04.2026 р. – 29.05.2026 р.	
9	Оформлення тексту роботи	01.06.2026 р. – 06.06.2026р.	
10	Попередній захист роботи на кафедрі	08.06.2026 р.	
11	Доопрацювання роботи з урахуванням зауважень і пропозицій	09.06.2026 р. – 10.06.2026 р.	
12	Нормоконтроль	11.06.2026 р. – 15.06.2026 р.	
13	Захист кваліфікаційної роботи	16.06.2026 р. - 18.06.2026 р.	

Здобувач вищої освіти **Іван ВОВНЯНКО**

Керівник роботи **Ігор СЛЮСАРЬ**

ЗМІСТ

ВСТУП ……..…. 6
РОЗДІЛ 1. ОСОБЛИВОСТІ ЗАСТОСУВАННЯ ФРЕЙМВОРКІВ
AI-АГЕНТІВ ….…….……. 8
1.1 Поняття агента та асистента на основі штучного інтелекту ….…… 8
1.2 Особливості реалізації фреймворків AI-агентів ….……...……. 10
1.3 Аналіз обмежень застосування фреймворків AI-агентів ……..……. 13
1.4 Тенденції розвитку агентних систем ……..……..……... 19
РОЗДІЛ 2. ВИКОРИСТАННЯ ФРЕЙМВОРКІВ ДЛЯ ПОБУДОВИ
AI-АГЕНТА …….…….…….……..……..……. 21
2.1 Систематизація існуючої номенклатури фреймворків AI-агентів …… 21
2.2 Визначення послідовності впровадження agentic AI у
робочий процес ……..…….…….…….…….…….……. 28
2.3 Узагальнена класифікація типових архітектур фреймворків
AI-агентів ……..…….……..…….…….…….…….…… 31
2.4 Реалізація локального AI-агента …….…….…….……... 36
2.5 Застосування Docker для локального AI-агента ……..…….…….. 44
РОЗДІЛ 3. РЕКОМЕНДАЦІЇ ЩОДО ПРАКТИЧНОЇ РЕАЛІЗАЦІЇ
AI-АГЕНТА ……. 46
3.1 Експериментальна реалізація AI-агента ……. 46
3.2 Формування структурованих інструкцій для AI-агентів ….……. 49
3.3 Типи агентів та їх можливості ……..…….…….…….…….…… 53
3.4 Рекомендації з вибору інструментарію для реалізації AI-агентів ……. 56
3.5 Економічне обґрунтування прийнятих рішень ……... 60
ВИСНОВКИ ……..…….. 63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ……... 65
ДОДАТКИ ……..……. 69

ВСТУП

Актуальність теми кваліфікаційної роботи підтверджується необхідністю застосування AI-агентів в якості інструменту для автоматизації складних інформаційних та бізнес-процесів. Їх особливістю є можливість локального розгортання в інтересах зменшення залежності від хмарних сервісів для обробки конфіденційних даних. Фреймворки AI-агентів спрощують процес побудови агентних систем за рахунок надання готових компонентів для взаємодії з мовними моделями, інструментами, пам’яттю та механізмами планування. Це дозволяє розробнику не створювати всю логіку роботи AI-агента з нуля, а використовувати вже підготовлені рівні абстракції. Однак, велика кількість абстрактних шарів може ускладнювати розуміння внутрішньої логіки роботи системи та її налагодження. Тому під час вибору фреймворку важливо враховувати не лише його функціональні можливості, а й простоту використання, прозорість архітектури та придатність до конкретного завдання. Все це свідчить про актуальність теми роботи.

Метою кваліфікаційної роботи є підвищення ефективності створення AI-агентів за рахунок застосування фреймворків, які забезпечують побудову агентної логіки, інтеграцію мовних моделей та практичну реалізацію локальних AI-агентів.

Завданнями кваліфікаційної роботи є:

– дослідити особливості застосування та тенденції розвитку фреймворків AI-агентів;

– провести систематизацію сучасних фреймворків AI-агентів і узагальнити типові архітектури їх побудови;

– визначити послідовність впровадження agentic AI у робочий процес та особливості формування структурованих інструкцій для AI-агентів;

– розробити рекомендації щодо вибору інструментарію та практичної реалізації AI-агентів.

Об'єктом дослідження є процес створення та практичного застосування AI-агентів на основі сучасних фреймворків AI.

Предметом дослідження є фреймворки, архітектурні підходи, програмні засоби та методи синтезу AI-агента.

Методами дослідження є в рамках визначення особливостей застосування фреймворків AI-агентів та економічного оцінювання доцільності прийнятих рішень використовувався аналітичний метод досліджень; для зіставлення сучасних фреймворків AI-агентів, типових архітектурних підходів та варіантів їх практичного використання – порівняльний аналіз; побудови структури локального AI-агента та визначення послідовності його впровадження у робочий процес – моделювання.

Інформаційна база кваліфікаційної роботи сформована з Інтернет-ресурсів, що місять інформацію про LLM, AI-агенти, agentic AI, фреймворки AI-агентів.

Практична значущість роботи полягає в узагальненій класифікації типових архітектур фреймворків AI-агентів, розробці моделі локального AI-агента на основі PicoClaw та рекомендацій щодо практичної реалізації реалізації AI-агента. Вони можуть бути використані для подальших досліджень за даною тематикою та при проектуванні локальних AI-агентів.

Апробація результатів відбувалася в рамках XXI щорічної студентської наукової конференції «Сучасні інформаційні технології та інноваційні методики в економіці, менеджменті та бізнесі» Полтавського державного аграрного університету (22 квітня 2026 р., м. Полтава).

За результатами досліджень здійснено публікацію тез доповідей.

Структура кваліфікаційної роботи логічно пов'язана з завданнями досліджень і містить вступ, три розділи основної частини, висновки, список використаних джерел, додатки. Загальний обсяг пояснювальної записки кваліфікаційної роботи складає 69 сторінок формату А4. Вона містить 26 рисунків і 2 таблиці.

РОЗДІЛ 1
ОСОБЛИВОСТІ ЗАСТОСУВАННЯ ФРЕЙМВОРКІВ AI-АГЕНТІВ

1.1 Поняття агента та асистента на основі штучного інтелекту

На даний час, на основі штучного інтелекту (Artificial Intelligence, AI) може створюватись програмна система, яка здатна самостійно виконувати певні дії для досягнення поставленої мети, адаптуватись до ситуації та вибору оптимального способу дії [1]. Зазвичай, її називають AI-агентом [2]. Він може отримувати вхідні дані, аналізувати їх, визначати послідовність подальших кроків, використовувати додаткові інструменти та формувати результат. У загальному вигляді робота AI-агента починається з отримання вхідних даних. Це може бути текстовий запит користувача, голосове повідомлення, файл, дані з вебсайту, інформація з БД або результат роботи іншої програми. Після цього AI-агент виконує аналіз отриманої інформації: визначає зміст запиту, виділяє ключові дані, встановлює намір користувача та формує внутрішнє уявлення про задачу, яку потрібно вирішити. Наступним етапом є планування дій. AI-агент може визначати послідовність кроків, необхідних для виконання завдання. Наприклад, якщо користувач просить підготувати відповідь на звернення, агент спочатку аналізує зміст звернення, потім шукає необхідну інформацію, формує відповідь і за потреби передає результат людині для перевірки. Саме здатність розбивати завдання на окремі етапи є однією з головних особливостей AI-агента. Важливу роль у роботі AI-агента відіграє мовна модель (Large Language Models, LLM) [3]. Вона забезпечує розуміння природної мови та генерацію відповідей. Завдяки цьому агент може взаємодіяти з користувачем у звичній текстовій або голосовій формі. Однак сам AI-агент не обмежується лише LLM (рис.1.1). До його складу можуть входити модулі пам’яті, планування, інструменти для роботи з файлами, БД, API, вебсервісами або локальними програмами. Завдяки цьому AI-агент можна розглядати як більш складну систему, яка поєднує

можливості AI та автоматизації окремих бізнес-процесів. Тобто сутність AI-агента полягає у поєднанні аналітичної та виконавчої функцій (може виконувати конкретні дії). У такому випадку AI-агент виступає як інтелектуальний помічник, який зменшує навантаження на людину та пришвидшує обробку інформації.

Рисунок 1.1 – Спрощена структура AI-агента

AI-агент може працювати як у хмарному середовищі, так і локально. Локальне розгортання є особливо важливим у випадках, коли обробляються конфіденційні дані. У такій архітектурі всі основні процеси виконуються на локальному сервері або комп’ютері без передавання даних до сторонніх сервісів. Це підвищує рівень контролю над інформацією та зменшує ризики витоку даних. AI-агент може працювати за певним сценарієм або логікою, яка задається розробником. Він може мати роль, наприклад агента технічної підтримки, аналітика звернень, помічника для обробки документів або агента для автоматизації бізнес-процесів. У межах цієї ролі AI-агент виконує не одну окрему відповідь, а цілий набір пов’язаних дій.

Існує поняття AI-асистенту. Його основна функція полягає в інформаційній підтримці людини. Він переважно виконує роль помічника для користувача. Наприклад, відповідає на запитання, пояснює інформацію, допомагає написати текст, перекласти фразу, знайти ідею або підготувати певний матеріал. Тобто AI-асистент здебільшого працює у форматі діалогу: користувач ставить запитання або дає команду, а система формує відповідь. Головна різниця між AI-агентом та AI-асистентом полягає у рівні самостійності та здатності виконувати дії. AI-асистент переважно реагує на запити користувача та допомагає у вирішенні окремих завдань. AI-агент може мати певну мету, планувати послідовність дій і взаємодіяти з іншими

програмними компонентами для досягнення результату. Наприклад, асистент може підказати, як написати відповідь на звернення клієнта, а AI-агент може самостійно прийняти звернення, класифікувати його, знайти потрібні дані, сформувати відповідь і передати її у відповідну систему. Ще однією відмінністю є наявність інструментів. AI-асистент часто обмежується генерацією текстової відповіді, тоді як AI-агент може використовувати зовнішні або локальні інструменти. Наприклад, він може запускати скрипти, звертатися до БД, аналізувати документи, створювати файли, працювати з електронною поштою або виконувати дії в інформаційній системі. Саме ця можливість робить AI-агента корисним для автоматизації робочих процесів.

1.2 Особливості реалізації фреймворків AI-агентів

Якщо AI-агент – це «розумний виконавець завдань», то фреймворк AI-агента – це середовище, яке допомагає цього виконавця побудувати без написання всієї логіки з нуля. Тобто фреймворк AI-агента – це програмна платформа або набір бібліотек, який надає готові компоненти для створення, налаштування та керування AI-агентами. Зазвичай, фреймворк реалізує: взаємодію з LLM; управління діалогом із користувачем; використання зовнішніх інструментів (API, БД, вебпошуку тощо); збереження пам’яті про попередні дії та повідомлення; планування та виконання послідовності кроків для досягнення поставленої мети; взаємодію між кількома агентами (у мультиагентних системах).

У фреймворка AI-агента є рівні (шари абстракції), які приховують складні технічні деталі роботи агента та дозволяють розробнику описувати його поведінку простішими поняттями (завдання, інструменти, ролі, пам’ять, дії, сценарії роботи). Інакше кажучи, фреймворк створює «надбудову» над LLM, щоб розробник не працював напряму лише з API моделі, а міг будувати повноцінного AI-агента. Тобто, абстракція – це приховування деталей

реалізації за простішим інтерфейсом. Фреймворк сам вирішує: яку модель викликати, коли звертатися до інструментів, як передати результат назад у модель, як зберегти контекст. У сучасних фреймворках нерідко з'являються кілька рівнів абстракції (рис. 1.2). Через це розробнику іноді складніше зрозуміти, що реально відбувається всередині.

Рисунок 1.2 – Багаторівнева абстракція у фреймворках

Зазвичай, у фреймворка є такі основні шари абстракції.

1. Шар мовної моделі. Це базовий рівень, на якому працює LLM. Вона відповідає за розуміння запиту користувача, генерацію тексту, аналіз інформації та формування відповідей. На цьому рівні можуть використовуватись як хмарні моделі, так і локальні моделі.

2. Шар агента описує самого AI-агента: його роль, мету, інструкції та правила поведінки. Наприклад, агент може бути «асистентом служби підтримки», «аналітиком звернень», «помічником для роботи з документами» або «агентом для автоматизації бізнес-процесів». Завдяки цьому розробнику не потрібно щоразу вручну пояснювати моделі, як вона має поводитися. Ці правила задаються у вигляді конфігурації AI-агента.

3. Шар інструментів. AI-агент часто повинен не лише відповідати текстом, а й виконувати дії. Для цього фреймворк надає шар інструментів.

Через нього агент може звертатися до БД, запускати Python-код, працювати з файлами, виконувати пошук, викликати API або взаємодіяти з іншими програмами. Наприклад, агент може отримати запит користувача, знайти потрібну інформацію в документі, проаналізувати її та сформувати відповідь.

4. Шар пам’яті відповідає за збереження контексту. Пам’ять дозволяє агенту враховувати попередні повідомлення, історію дій або додаткові дані про задачу. Без шару пам’яті агент працював би лише з поточним запитом. Із пам’яттю він може підтримувати більш послідовну взаємодію з користувачем.

5. Шар планування. На цьому рівні агент визначає, які кроки потрібно виконати. Наприклад, якщо користувач просить підготувати відповідь на звернення, агент може спочатку класифікувати звернення, потім знайти потрібні дані, сформувати короткий зміст і тільки після цього створити відповідь. Тобто шар планування дозволяє агенту не просто генерувати текст, а будувати послідовність дій.

6. Шар виконання дій відповідає за практичне виконання запланованих кроків. Наприклад, агент може викликати потрібний інструмент, передати дані в іншу систему, створити файл, оновити запис у таблиці або надіслати результат користувачу. Саме цей рівень перетворює AI-агента з простого чат-бота на систему автоматизації.

7. Шар взаємодії між агентами. У деяких фреймворках передбачена робота не одного, а кількох агентів. Наприклад, один агент може аналізувати запит, другий – шукати інформацію, третій – перевіряти результат, а четвертий – формувати фінальну відповідь. Такий підхід називають мультиагентною взаємодією. Він використовується тоді, коли складну задачу зручно розділити між кількома спеціалізованими агентами.

1.3 Аналіз обмежень застосування фреймворків AI-агентів

Перші фреймворки для AI-агентів з’явилися приблизно у 2022-2023 роках. Тоді вважалось, що LLM достатньо дати інструменти та ціль без – вона сама розбереться (агент думає, планує і діє). Кількість фреймворків різко зростала. Найбільш відомими з них стали LangChain [4] (перший мейнстрімний), AutoGen від к. Microsoft (мультиагентні діалоги) [5], CrewAI («команди» агентів із ролями, як у корпоративній оргструктурі) [6], LlamaIndex [7] (RAG, агенти, інструментарій для роботи з даними), Haystack [8], SuperAGI [9], AgentGPT [10] та ще понад двадцять назв. Популяризація концепції AI-агентів сприяла появі значної кількості спеціалізованих фреймворків та програмних платформ. Демонстраційні приклади показують можливість автономного виконання агентами багатоетапних завдань, включаючи аналіз інформації, прийняття рішень, використання зовнішніх інструментів і автоматизацію процесів розробки програмного забезпечення.

Попри значний інтерес до агентних систем та швидке поширення відповідних фреймворків, практичне використання AI-агентів виявило низку суттєвих проблем і обмежень.

Першою проблемою є недостатня надійність роботи агентів. На відміну від традиційних програмних систем із детермінованою логікою, поведінка AI-агента значною мірою залежить від результатів генерації LLM. Це може призводити до різних результатів під час багаторазового виконання однакових завдань. У деяких випадках агенти можуть некоректно використовувати інструменти, потрапляти в циклічні послідовності дій або формувати помилкові висновки, що негативно впливає на стабільність системи.

Другою проблемою є висока вартість обчислювальних ресурсів. Виконання складних агентних сценаріїв часто потребує багаторазових звернень до LLM та зовнішніх сервісів. У разі неефективного планування дій або виникнення циклічних процесів витрати на обробку запитів можуть суттєво перевищувати очікувані значення.

Третім викликом є складність моніторингу та аналізу роботи агентів. Під час виконання багатоетапних завдань не завжди легко визначити причини прийняття того чи іншого рішення, вибору конкретного інструмента або формування певної відповіді. Це ускладнює процес тестування, налагодження та забезпечення якості функціонування агентної системи.

Окремої уваги заслуговує проблема надмірної складності деяких агентних фреймворків. Наприклад, популярний фреймворк LangChain швидко розвивався та постійно розширював функціональні можливості, що призвело до появи великої кількості рівнів абстракції та частих змін програмних інтерфейсів. У результаті розробка та супровід проєктів на його основі могли вимагати додаткових зусиль порівняно з більш простими або спеціалізованими рішеннями.

Подальший розвиток агентних фреймворків привів до появи нових підходів, спрямованих на підвищення керованості та передбачуваності роботи агентів. Одним з таких рішень став LangGraph [4], представлений у 2024 р. як розширення екосистеми LangChain. Проте навіть поява таких рішень не усунула повністю проблеми, які характерні для агентних систем. Складність архітектури, потреба в додатковому налаштуванні та залежність від поведінки мовних моделей залишаються актуальними викликами.

Це свідчить про те, що розвиток фреймворків AI-агентів продовжується, а пошук балансу між функціональністю, керованістю та простотою використання залишається одним пріоритетних напрямів досліджень у цій галузі.

Практика використання агентних систем показала, що після початкового етапу активного зростання інтересу до AI-агентів відбулося певне переосмислення їхньої ролі. Ринок не припинив розвиток, однак поступово відійшов від надмірно складних універсальних рішень у бік більш простих і практично орієнтованих підходів. Одним із важливих висновків стало те, що найбільш життєздатною виявилася не конкретна реалізація у вигляді певного фреймворку, а сам агентний підхід. Ідея поетапного

виконання завдань, використання LLM для прийняття рішень, звернення до зовнішніх інструментів і перевірки результату є ефективною. Водночас багато готових агентних фреймворків виявилися надто складними для типових практичних задач. У багатьох випадках замість використання великих фреймворків доцільнішим є створення простішої агентної логіки, побудованої за принципом послідовного виконання кроків. Такий підхід нагадує кінцевий автомат, у якому розробник явно задає можливі стани системи та правила переходу між ними.

Наприклад, без використання LangChain або CrewAI [6], просто: prompt → tool call → check result → next step. Завдяки цьому система стає більш зрозумілою, керованою та зручною для налагодження.

Окремо слід зазначити, деяка частина завдань, які спочатку намагалися розв’язувати за допомогою складних агентних ланцюжків, на практиці ефективно вирішується за допомогою пошуково-підкріпленої генерації (Retrieval-Augmented Generation, RAG) [11]. Поєднання мовної моделі з пошуком релевантної інформації в базі знань часто забезпечує більш стабільний і передбачуваний результат, ніж багатоетапна автономна робота агента. Тобто, AI-агент здавався потужнішим, але RAG працював надійніше.

Важливу роль також відіграє структурований вивід даних. Механізми на зразок function calling або JSON mode дозволяють моделі формувати відповідь у заздалегідь визначеному форматі [12]. Це зменшує потребу в окремих агентних сценаріях, де раніше агент самостійно визначав структуру результату. У таких випадках завдання може бути реалізоване простіше: модель одразу повертає дані у потрібній структурі, що полегшує їх подальшу обробку програмними засобами.

За період активного розвитку AI-агентів було виявлено низку типових помилок, які часто виникають під час проєктування агентних систем.

Однією з таких помилок є надмірна багаторівнева оркестрація [13]. Вона виникає тоді, коли один агент керує іншими агентами, які, у свою чергу, також керують окремими підсистемами або виконавцями. Така архітектура

може виглядати логічною на схемі, однак на практиці вона часто призводить до ускладнення контролю виконання, накопичення помилок і збільшення часу обробки запитів.

Іншою поширеною проблемою є створення AI-агента з надмірною кількістю інструментів. Якщо один AI-агент має доступ до великого набору функцій, API або сервісів LLM може некоректно обирати потрібний інструмент або використовувати його не за призначенням. Тому доцільно обмежувати кількість інструментів для одного AI-агента та, за потреби, створювати кілька спеціалізованих AI-агентів із чітко визначеними ролями.

Ще однією помилкою є відсутність перевірки результатів роботи інструментів. У простих демонстраційних прикладах AI-агент може одразу переходити до наступного кроку після отримання відповіді від зовнішнього сервісу. Проте в реальних системах необхідно передбачати валідацію результатів, обробку помилок, повторні спроби виконання запиту та механізми аварійного завершення сценарію.

Також важливою проблемою є недостатнє управління станом агентної системи. Якщо AI-агент не зберігає інформацію про попередні дії, проміжні результати та поточний етап виконання задачі, він може втрачати контекст або повторювати вже виконані кроки. Тому для складних агентних сценаріїв необхідно реалізовувати механізми збереження стану, історії дій та контролю переходів між етапами виконання завдання.

Незважаючи на наявні обмеження та складнощі, агентні системи вже сьогодні успішно застосовуються в багатьох практичних сферах. Найбільшу ефективність вони демонструють у задачах, де можна чітко визначити послідовність дій, контролювати результати виконання та обмежити область відповідальності агента.

Одним із найбільш успішних напрямів є підтримка процесу розробки ПЗ. Сучасні AI-асистенти здатні аналізувати програмний код, виявляти потенційні помилки, пропонувати варіанти виправлення та автоматично генерувати окремі фрагменти програм. Такі системи фактично реалізують

агентний підхід, поєднуючи мовні моделі з інструментами роботи з кодом, системами контролю версій та середовищами розробки.

Широке застосування агенти знаходять і в задачах обробки документів. Вони можуть автоматично аналізувати договори, рахунки, звіти та інші документи, вилучати необхідні дані та передавати їх до інформаційних систем. Надійність таких рішень забезпечується можливістю автоматичної або ручної перевірки отриманих результатів.

Перспективним напрямом є використання агентів у системах підтримки користувачів першого рівня. У таких системах агент відповідає на типові запитання, надає довідкову інформацію або допомагає виконати стандартні операції. При цьому область його відповідальності зазвичай чітко обмежується, а складні або нестандартні звернення автоматично передаються оператору для подальшого опрацювання.

Агентні технології застосовуються в системах моніторингу та аналізу подій. Агент може збирати інформацію з журналів подій, системних метрик або інших джерел даних, формувати узагальнені звіти, виявляти потенційні проблеми та пропонувати можливі дії для їх усунення. Остаточне рішення при цьому приймає людина-оператор.

Окреме місце займають корпоративні системи пошуку знань та підтримки прийняття рішень. Такі рішення часто базуються на поєднанні агентного підходу з RAG. Агент отримує доступ до внутрішньої бази знань організації, виконує пошук релевантної інформації та формує відповіді на запити користувачів. Завдяки використанню актуальних даних підприємства такі системи демонструють високу практичну ефективність і вже активно впроваджуються в сучасних організаціях.

Як наслідок, не всі фреймворки для створення AI-агентів змогли зберегти популярність і актуальність у процесі розвитку галузі. Частина рішень виявилася надто складною для практичного використання або не змогла забезпечити необхідний рівень надійності та масштабованості. Водночас окремі платформи поступово зайняли стійкі позиції та сформували

власні напрями застосування. Серед сучасних фреймворків особливу увагу привертають LangGraph та LlamaIndex. Ці інструменти орієнтовані на вирішення конкретних класів задач і активно використовуються в реальних проєктах. Наприклад, LangGraph – якщо потрібні реальні агентні сценарії з явним графом переходів та людським контролем (складно налаштовувати, але працює передбачувано), LlamaIndex [7] – якщо завдання пов'язане з обробкою даних (складний RAG, робота з документами, пошук корпоративної бази знань) – в цьому сегменті поки що не має реальних конкурентів.

Важливим етапом розвитку агентних технологій стала поява протоколу Model Context Protocol (MCP) – рис. 1.3. Його основною метою є стандартизація взаємодії між LLM та зовнішніми інструментами. До появи MCP різні фреймворки використовували власні механізми інтеграції, що ускладнювало сумісність між рішеннями [14]. Використання єдиного протоколу дозволяє розробляти інструменти один раз і застосовувати їх у різних агентних екосистемах. Така уніфікація сприяє розвитку спільної інфраструктури та спрощує створення нових агентних систем.

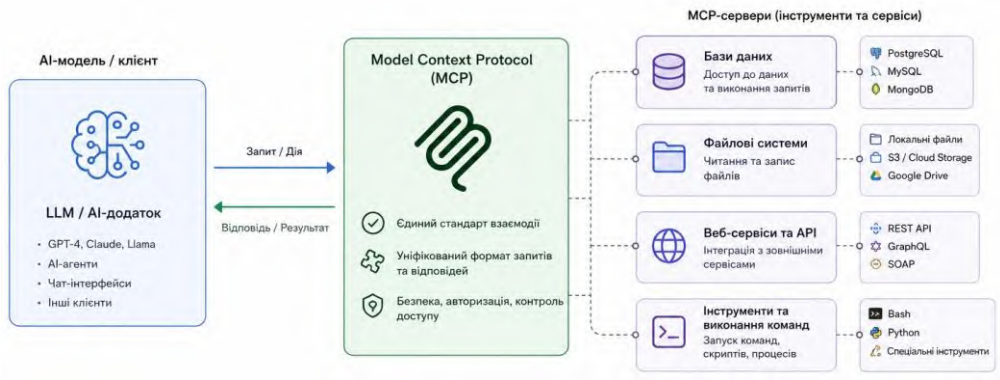

Рисунок 1.3 – Протокол MCP

Окремої уваги заслуговує розвиток локальних AI-агентів. З поширенням локальних LLM і платформ для їх запуску стало можливим створення агентних систем без використання хмарних сервісів. Саме тому локальне розгортання агентних систем сьогодні розглядається як тренд розвитку технологій AI.

1.4 Тенденції розвитку агентних систем

Крім визначення AI-агент існує ще поняття agentic AI [15]. Це система або архітектурний підхід, у межах якого один або декілька AI-агентів можуть взаємодіяти між собою, розподіляти підзавдання, використовувати пам’ять, інструменти та механізми оркестрації для досягнення складнішої мети. Agentic AI часто розглядається як розвиток і розширення ідеї AI-агентів [16].

Аналіз сучасного стану розвитку agentic AI дозволяє визначити кілька ключових трендів, які, ймовірно, будуть визначати подальший розвиток цієї галузі. Однією з найбільш помітних тенденцій є перехід від універсальних агентів до спеціалізованих рішень. Якщо на початкових етапах розвитку значна увага приділялася створенню універсальних агентів, здатних виконувати широкий спектр завдань, то сьогодні перевага все частіше надається вузькоспеціалізованим системам. Такі агенти розробляються для виконання конкретних функцій, наприклад програмування, аналізу даних, обробки документів або підтримки користувачів. Чітко визначена область відповідальності дозволяє підвищити якість роботи та спростити контроль результатів. Іншим перспективним напрямом є розвиток технологій Computer Use [17], які дозволяють агентам взаємодіяти з графічними інтерфейсами програм і вебресурсів. Такі системи можуть працювати з браузерами, настільними додатками та іншими програмними засобами аналогічно до дій людини. Хоча на сьогодні подібні рішення ще мають обмеження щодо надійності та точності виконання операцій, темпи їх розвитку свідчать про значний потенціал для практичного застосування. Важливою тенденцією також є поступовий перехід від великої кількості несумісних між собою фреймворків до стандартизації протоколів взаємодії, наприклад, MCP. Стандартизація сприяє підвищенню сумісності різних рішень і спрощує розробку агентних систем. Окремий інтерес викликає розвиток локальних та периферійних (Edge) agentic AI. Підвищення якості компактних LLM і зростання обчислювальних можливостей локальних пристроїв створюють

умови для розгортання AI-агентів без використання хмарної інфраструктури. Це актуально для корпоративних інформаційних систем, пов’язаних з конфіденційними даними та з обмеженим доступом до Інтернет.

Таким чином, розвиток агентних технологій демонструє поступовий перехід від етапу активного інтересу та експериментів до етапу практичного впровадження. Досвід останніх років показав, що найбільш ефективними виявляються рішення, які поєднують функціональність із простотою реалізації та контролем роботи системи. Сучасні підходи до розробки AI-агентів орієнтовані на підвищення надійності, прозорості та керованості їхньої поведінки. Важливим результатом еволюції agentic AI стало усвідомлення необхідності використання агентного підходу лише там, де він дійсно забезпечує переваги порівняно з традиційними програмними рішеннями. Для багатьох задач достатньо використання детермінованих алгоритмів або простих сценаріїв автоматизації, тоді як агентні технології доцільно застосовувати для складних багатоетапних процесів, що потребують аналізу інформації та прийняття рішень. сучасні фреймворки та інструменти забезпечують кращі можливості для моніторингу, аналізу та налагодження роботи агентів. Це дозволяє зменшити рівень невизначеності та підвищити довіру до результатів функціонування системи. Водночас у критично важливих сферах застосування зберігається необхідність участі людини в процесі прийняття остаточних рішень. Такий підхід дозволяє поєднати переваги автоматизації з контролем з боку фахівця, що сприяє підвищенню надійності та безпечності використання агентних технологій.

РОЗДІЛ 2
ВИКОРИСТАННЯ ФРЕЙМВОРКІВ ДЛЯ ПОБУДОВИ AI-АГЕНТА

2.1 Систематизація існуючої номенклатури фреймворків AI-агентів

На даний час, для створення AI-агентів існує велика номенклатура фреймворків у вигляді окремих інструментів, IDE та платформ agentic AI. Тому доцільно дослідити їх сфери їх застосування та особливості. Під час аналізу таких рішень варто враховувати кілька критеріїв. По-перше, сучасні AI-агенти мають не лише генерувати відповіді, а й виконувати багатоступінчасте планування, використовувати інструменти, плагіни, API або взаємодіяти з інтерфейсом користувача. По-друге, важливим є їх реальне практичне застосування, тобто здатність автоматизувати робочі процеси, а не лише виконувати прості текстові запити. По-третє, слід оцінювати широту використання таких платформ у різних сферах: розробці програмного забезпечення, маркетингу, операційній діяльності, дослідженнях, продажах, аналізі даних та інших напрямах. Окрему увагу варто приділити рівню впровадження та розвитку таких інструментів: наявності активної спільноти, регулярних оновлень, документації та прикладів використання. Це дозволяє відрізнити реально придатні до роботи рішення від демонстраційних або експериментальних проєктів.

1. Фреймворки та платформи agentic AI загального призначення. До першої групи можна віднести фреймворки та платформи agentic AI загального призначення. Вони виконують роль універсальних інструментів для створення AI-агентів, оскільки дають змогу планувати послідовність дій, підключати зовнішні інструменти, працювати з API та організовувати виконання складних завдань. Серед таких рішень можна виділити OpenAI GPTs та Assistants API. Їх agentic-можливості полягають у підтримці виклику інструментів, роботі з файлами, отриманні інформації, використанні інтерпретатора коду та збереженні контексту взаємодії. Такі рішення

доцільно застосовувати для створення прототипів персональних або командних помічників, які можуть аналізувати дані, звертатися до зовнішніх сервісів і підтримувати контекст роботи. Водночас варто враховувати обмеження швидкості та можливе зростання витрат у разі активного використання зовнішніх викликів. Anthropic Claude Projects & Workflows орієнтовані на роботу з довгими документами, аналітичними завданнями та побудову послідовних робочих процесів [18]. Їх перевагою є здатність до якісного аналізу й міркування, що робить їх корисними для дослідницьких і корпоративних завдань. При цьому використання інструментів у таких системах може бути більш обмеженим, тому запити доцільно формулювати як чітке технічне або проєктне завдання. Google Gemma [19] / Vertex AI Agents [20] забезпечують оркестрацію, обґрунтування дій та інтеграцію інструментів у межах Google Cloud. Такі рішення найкраще підходять для команд, які вже використовують BigQuery, Google Cloud Storage, Google Docs та інші сервіси Google. Недоліком може бути складність початкового налаштування. Microsoft Copilot Studio дозволяє створювати агентів, інтегрованих із корпоративними даними та сервісами Microsoft 365, Power Automate і Dynamics [21]. Це рішення доцільне для організацій, які активно використовують Outlook, Teams, SharePoint та інші продукти Microsoft. Водночас важливим є правильне налаштування доступів і дозволів, оскільки агент може виконувати дії з корпоративними даними. LangChain Agents є популярним фреймворком для Python та JavaScript, який підтримує використання інструментів, планування, пам’ять і моніторинг роботи агентів. Його доцільно застосовувати розробникам, які створюють власні agentic-системи. Однак через велику кількість абстракцій такі системи потребують ретельного тестування та відстеження виконання. LlamaIndex використовується для побудови агентів, які активно працюють із даними, документами та пошуком інформації. Він підтримує конектори до джерел даних, маршрутизацію запитів, структуровані відповіді та агентні цикли. Таке рішення є доцільним для систем, де важливими є пошук, посилання на

джерела та подальші дії з інформацією. Водночас ефективність роботи значною мірою залежить від правильної стратегії індексування даних. AutoGen орієнтований на створення багатоагентних систем, у яких кілька агентів взаємодіють між собою, розподіляють завдання та узгоджують результати. Це корисно для складних інженерних робочих процесів, що поєднують аналіз даних, програмування та перевірку якості. При цьому велика кількість агентів може ускладнювати керування системою, тому доцільним є використання агента-супервізора. CrewAI також підтримує багатоагентний підхід, але робить акцент на ролях агентів, плануванні, делегуванні та перевірці результатів. Його можна застосовувати для контентних, аналітичних і дослідницьких процесів. Для ефективної роботи необхідно чітко визначати ролі агентів, їхні завдання та критерії успішності. Flowise є візуальним конструктором для створення агентів і робочих процесів у стилі LangChain. Він підходить для користувачів без глибоких навичок програмування, але хочуть швидко створювати прототипи агентних систем. Основним обмеженням є ризик надмірного ускладнення візуальних схем, тому такі потоки бажано проєктувати максимально просто. Zapier Central / AI Actions забезпечує керування автоматизаціями між великою кількістю вебзастосунків за допомогою природної мови [22]. Такі інструменти можуть об’єднувати електронну пошту, CRM-системи, таблиці, месенджери та інші сервіси в єдину агентну систему. Проте перед використанням у реальних бізнес-процесах необхідно передбачити запобіжники, тестові сценарії та контроль виконуваних дій.

2. Агенти для перегляду вебсторінок і виконання дій у браузері. Окрему групу становлять AI-агенти, які можуть виконувати веброботу замість користувача: переглядати сторінки, переходити за посиланнями, натискати елементи інтерфейсу, збирати інформацію та автоматизувати повторювані дії. Такий напрям часто пов’язують із концепцією Computer Use, тобто здатністю AI-системи взаємодіяти з браузером або програмним інтерфейсом подібно до людини. Perplexity Pages + Automations можна розглядати як інструмент для

дослідження інформації з подальшим структуруванням результатів у вигляді сторінок [23]. Його agentic-можливості полягають у пошуку даних із посиланнями на джерела та частковій автоматизації завдань. Найкраще такий підхід підходить для швидкої підготовки дослідницьких оглядів і коротких аналітичних матеріалів. Водночас потрібно перевіряти якість джерел, оскільки за недостатньої доказової бази система може формувати занадто впевнені висновки. Arc Search + Browse for Me орієнтований на збирання та узагальнення вебсторінок [24]. Його можна використовувати для попереднього аналізу інформації, швидкого огляду теми та формування коротких підсумків. Основним обмеженням є те, що система не завжди однаково добре відрізняє авторитетні джерела від менш надійних. Browse.ai призначений для моніторингу сайтів і вилучення структурованих даних. Він може «навчатися» на конкретній сторінці, збирати потрібну інформацію та відстежувати її зміни. Такий інструмент доцільно використовувати для моніторингу цін, збору потенційних клієнтів, оновлення каталогів або контролю змін на вебресурсах. При цьому слід враховувати, що зміна дизайну сайту може порушити роботу сценарію, тому іноді потрібне повторне налаштування. TaskMagic дозволяє записувати послідовність дій у браузері та автоматично відтворювати їх за допомогою AI. Це корисно для повторюваних вебзавдань, таких як завантаження файлів, заповнення форм, натискання кнопок або перенесення даних між сервісами. Обмеженням є залежність від стабільності інтерфейсу сайту: якщо елементи змінюються, автоматизований сценарій може працювати некоректно. UiPath Autopilot поєднує класичну роботизовану автоматизацію процесів із можливостями AI [25]. Це рішення орієнтоване на корпоративні сценарії, зокрема фінансові, операційні та адміністративні процеси, де важливі точність, контроль дій і ведення журналів. Водночас впровадження таких систем потребує часу, але є доцільним у випадках великого обсягу повторюваних операцій.

3. Агенти для коду, даних та інженерії. Окрему групу становлять AI-агенти, призначені для роботи з програмним кодом, даними та

інженерними завданнями. Вони можуть допомагати у плануванні змін, редагуванні файлів, запуску тестів, аналізі даних, створенні SQL-запитів і побудові прототипів програмних рішень. GitHub Copilot Workspace має agentic-можливості завдяки тому, що може планувати зміни у коді, редагувати файли, запускати тести та створювати pull request [26]. Найкраще він підходить для прискорення розробки нового функціоналу та виправлення помилок. Водночас результати його роботи потребують обов'язкової перевірки з боку розробника. OpenDevin / SWE-bench agents є дослідницькими проєктами, які демонструють можливості автономного виправлення помилок у програмних репозиторіях. Вони є корисними для команд, що вивчають напівавтономну розробку програмного забезпечення. Однак для промислового використання такі рішення потребують додаткових механізмів контролю та безпеки. Databricks Genie Agents орієнтовані на роботу з даними та аналітикою. Вони можуть створювати конвеєри обробки даних, SQL-запити та інформаційні панелі за допомогою природної мови. Такі агенти доцільні для аналітичних команд, які часто працюють із великою кількістю спеціальних запитів. При цьому важливим залишається правильне управління даними та контроль їхньої якості. Snowflake Cortex Analyst забезпечує розмовний аналіз даних і виконання дій у середовищі Snowflake. Це рішення підходить для BI-команд, які працюють із SQL-запитами та бізнес-аналітикою. Водночас необхідно заздалегідь визначати єдині правила обчислення метрик, щоб уникнути розбіжностей у результатах аналізу. Replit Agents дозволяють писати, запускати та виправляти кодові проєкти безпосередньо у браузері. Вони зручні для швидкого створення прототипів, навчальних проєктів і невеликих застосунків. Однак під час їх використання варто задавати обмеження щодо часу та ресурсів, щоб уникнути неефективного виконання повторюваних дій.

4. Автоматизатори для продажів, маркетингу та операційної діяльності. Зазвичай, вони орієнтовані на продажі, маркетинг і операційні процеси. Вони можуть допомагати у пошуку потенційних клієнтів, підготовці листів,

оновленні CRM-систем, створенні контенту, плануванні кампаній та автоматизації внутрішніх робочих процесів. HubSpot AI Agents мають agentic-можливості завдяки пошуку лідів, створенню послідовностей електронних листів, оновленню CRM-записів і виконанню подальших дій. Таке рішення оптимальне для малих і середніх компаній, яким потрібна автоматизація частини процесів продажу. Водночас для якісної персоналізації необхідні реальні дані про клієнтів, інакше комунікація може виглядати шаблонною. Salesforce Einstein Copilot for Sales/Service може створювати відповіді, оновлювати записи та запускати заздалегідь визначені сценарії роботи. Це рішення доцільне для великих організацій, які вже активно використовують екосистему Salesforce. Перед масштабним впровадженням такі інструменти бажано тестувати в окремому середовищі. Jasper Campaigns орієнтований на маркетингові завдання: планування контенту для різних каналів, створення матеріалів і підготовку публікацій. Він підходить для маркетингових команд, які одночасно працюють із великою кількістю кампаній і каналів комунікації. Водночас важливо налаштувати тон і стиль бренду, щоб згенерований контент відповідав корпоративній комунікації. Notion AI Projects + Q&A забезпечує узагальнення документів, створення завдань і зв’язування сторінок у межах робочого простору Notion. Такий інструмент корисний для менеджерів проєктів і команд, які ведуть документацію та завдання в Notion. Однак сформовані зв’язки між сторінками й документами потрібно перевіряти, оскільки система може робити неточні припущення. Airtable AI з автоматизацією дозволяє класифікувати дані, вилучати інформацію та запускати багатокрокові автоматизації. Його доцільно використовувати для операційних процесів, пов’язаних із контентом, інвентаризацією, рекрутингом або внутрішніми базами даних. Для якісної роботи таких сценаріїв важливо попередньо структурувати й очистити дані.

5. Агенти для досліджень, письма та роботи зі знаннями. Ще одну групу становлять AI-агенти, призначені для досліджень, підготовки текстів і роботи з інформацією. Вони можуть виконувати багатокроковий пошук,

узагальнювати документи, працювати з джерелами, формувати структуровані результати та допомагати в організації знань. Perplexity Enterprise Pro має agentic-можливості завдяки підтримці багатокрокових запитів, роботі з цитуванням джерел і створенню дайджестів документів. Це рішення доцільно використовувати аналітикам, редакторам і дослідникам, яким важливо швидко отримувати інформацію з посиланнями на джерела. Водночас необхідно перевіряти актуальність даних, діапазони дат і різноманітність використаних джерел. Elicit є дослідницьким AI-агентом, орієнтованим на роботу з науковою літературою. Він може допомагати у пошуку релевантних статей, вилученні ключової інформації та поданні результатів у структурованому вигляді. Найкраще він підходить для академічних і маркетингових досліджень. При цьому слід враховувати наявність платних функцій або обмежень доступу. Агенти для дослідження лідів, подібні до Pipecandy, використовуються для збагачення даних про компанії, категоризації потенційних клієнтів і виявлення важливих бізнес-сигналів. Вони корисні для команд RevOps, продажів і маркетингу, які працюють із великими наборами клієнтських даних. Основним обмеженням є потреба в якісній структурі даних і підтриманні актуальних схем. Agentive-асистенти для електронної пошти в Gmail або Outlook можуть сортувати листи, створювати чернетки відповідей, планувати подальші дії та відстежувати важливі повідомлення. Такі інструменти підходять для зменшення навантаження під час роботи з електронною поштою та підтримання впорядкованої вхідної скриньки. Водночас на початковому етапі доцільно обмежити права на автоматичне надсилання повідомлень, доки система не буде достатньо перевірена.

Надалі виділимо основні переваги та недоліки розглянутого інструментарію для реалізації AI-агентів. До основних переваг належить економія часу, оскільки AI-агенти можуть виконувати повторювані або багатокрокові завдання замість користувача. Також важливою є узгодженість дій: агент працює за заданою логікою і не пропускає окремі етапи процесу.

Хороші платформи також забезпечують відстежуваність, тобто дають змогу переглядати журнали виконаних дій. Крім того, після правильного налаштування один і той самий робочий процес можна масштабувати для великої кількості завдань або користувачів. Водночас існують і недоліки. Початкове налаштування агентів може бути складним і потребувати часу. Також можливі проблеми з точністю, оскільки інтерфейси сервісів змінюються, а структури даних можуть оновлюватися. Окрему увагу слід приділяти вартості, адже використання API, зовнішніх інструментів і великої кількості запитів може поступово збільшувати витрати. Ще одним ризиком є надмірна автоматизація, коли система швидко виконує дії, але сам процес або результат є помилковим. Для зменшення цих ризиків доцільно починати з одного важливого робочого процесу, протестувати його, задокументувати логіку роботи, додати засоби моніторингу та лише після цього масштабувати рішення.

2.2 Визначення послідовності впровадження agentic AI у робочий процес

Під час першого впровадження agentic AI доцільно починати не з масштабної автоматизації всієї діяльності організації, а з одного конкретного робочого процесу, який має зрозумілу практичну цінність. Такий підхід зменшує ризики, простіше оцінити ефективність системи та виявити можливі помилки ще на початковому етапі.

Першим кроком є вибір одного робочого процесу з вимірюваною цінністю. Це може бути, наприклад, обробка звернень користувачів, підготовка коротких звітів, класифікація повідомлень, аналіз документів або автоматизація повторюваних адміністративних дій. Важливо, щоб результат такого процесу можна було оцінити за конкретними показниками: зекономлений час, кількість оброблених запитів, точність відповідей або

зменшення кількості ручних операцій. Далі необхідно визначити вхідні дані, очікувані результати, інструменти та можливі нестандартні ситуації. Тобто описують, які дані отримує AI-агент, які дії він має виконувати, з якими сервісами або базами даних взаємодіє та які результати повинен сформувати. Окремо слід передбачити крайні випадки: некоректні вхідні дані, відсутність потрібної інформації, помилки доступу, дублювання запитів або ситуації, у яких агент не повинен діяти автономно.

Третім кроком є вибір платформи, яка найкраще відповідає розташуванню та характеру даних. Якщо організація працює з конфіденційною інформацією, перевагу доцільно надавати рішенням, які можуть бути розгорнуті локально або в контрольованому середовищі. Це зменшує ризики передавання даних стороннім сервісам.

Далі потрібно підготувати покрокову специфікацію роботи AI-агента. Вона має бути чіткою, структурованою та практичною. Доцільно описувати логіку роботи у вигляді коротких пунктів: що агент отримує на вході, які перевірки виконує, які інструменти використовує, коли зупиняється, коли звертається до людини та який результат формує. Надмірно загальні або нечіткі формулювання можуть призвести до помилок у роботі системи.

Наступним етапом є створення тестового середовища, або «пісочниці». На цьому етапі AI-агент не повинен мати права змінювати реальні дані, надсилати повідомлення від імені користувача або виконувати дії, які можуть вплинути на робочі процеси організації. Такий режим дозволяє безпечно перевірити логіку роботи системи та виявити недоліки без ризику для основної інфраструктури.

Далі йде визначення метрик успішності. До них можуть належати мінімальний поріг точності, кількість правильно оброблених запитів, час виконання завдання, рівень помилок, економія робочого часу або якість сформованих відповідей.

Перед повноцінним запуском доцільно виконати серію тестових або «тіньових» проходів. Наприклад, агент може обробити 10 реальних або

наближених до реальних сценаріїв, але без фактичного впливу на систему. Результати таких запусків порівнюються з очікуваними діями людини, після чого виправляються помилки в логіці, підказках, інтеграціях або правилах прийняття рішень.

Після успішного тестування можна поступово дозволити агенту виконувати записи або зміни лише для безпечних, неруйнівних дій. Наприклад, це створення чернетки, додавання тегу, формування внутрішньої нотатки або оновлення статусу, який легко скасувати. Критичні операції, зокрема фінансові дії, зовнішні відправлення або видалення даних, на цьому етапі мають залишатися під контролем людини.

Далі систему доцільно розгорнути для невеликої групи користувачів. Обмежене впровадження дозволяє зібрати практичні відгуки, оцінити зручність використання, виявити нетипові сценарії та зрозуміти, як агент поводиться в реальних умовах. На основі цих відгуків можна вдосконалити інструкції, інтерфейс, правила доступу та механізми контролю.

Останнім кроком є налаштування сповіщень, регулярне оновлення ключів доступу та документування механізму аварійного вимкнення. Система повинна повідомляти відповідальних осіб про помилки, перевищення бюджету, незвичну активність або невдалі виконання сценаріїв. Також необхідно передбачити простий спосіб швидкого вимкнення агента у випадку некоректної роботи. Документація має містити опис логіки роботи, прав доступу, відповідальних осіб, порядку оновлення ключів і дій у разі виникнення інциденту.

Перший запуск agentic AI має бути поступовим, контрольованим і добре задокументованим. Такий підхід не є складним з організаційної точки зору, однак саме він дозволяє безпечно перевірити можливості AI-агента, оцінити його практичну користь і підготувати основу для подальшого масштабування автоматизації.

2.3 Узагальнена класифікація типових архітектур фреймворків AI-агентів

Існують не одна, а кілька архітектур фреймворків AI-агентів. Їх можна класифікувати за тим, як агент планує дії, використовує інструменти, зберігає стан і взаємодіє з іншими агентами. Як наслідок, спираючись на пп. 2.1 і 2.2, визначимо типові архітектури фреймворків AI-агентів.

1. Одноагентна архітектура (рис. 2.1). Основним компонентами є: мовна модель; prompt / системна інструкція; набір інструментів; короткочасна або довготривала пам’ять; модуль відповіді користувачу. Це найпростіший варіант: один LLM-агент отримує завдання, аналізує його, викликає інструменти та формує відповідь. Такий підхід підходить для простих асистентів: чат-ботів, помічників для пошуку інформації, генерації тексту, обробки документів.

2. Tool-calling / ReAct-архітектура – агент не просто відповідає текстом, а циклічно виконує дії: міркує, обирає інструмент, отримує результат і продовжує роботу (рис. 2.2). Наприклад, агент може спочатку знайти файл, потім прочитати його, потім зробити висновок. Це одна з базових архітектур сучасних агентів, бо вона дозволяє LLM взаємодіяти із зовнішніми системами: API, базами даних, файлами, пошуком, Python-кодом тощо. MCP також рухається саме в цьому напрямі: він стандартизує підключення AI-додатків до зовнішніх інструментів, джерел даних і workflow-систем [27].

Рисунок 2.1 – Одноагентна архітектура

Рисунок 2.2 – Tool-calling / ReAct-архітектура

3. Workflow-архітектура (рис. 2.3). Це більш контрольований підхід, де агент не діє повністю вільно, а виконує заздалегідь заданий сценарій. Такі архітектури характерні для автоматизації бізнес-процесів. Наприклад: отримати повідомлення → класифікувати → згенерувати відповідь → записати в таблицю → надіслати користувачу. Перевага workflow-архітектури – передбачуваність. Недолік – менша гнучкість, бо агент обмежений заданою логікою.

4. Graph-архітектура – робота агента подається як граф: вузли відповідають окремим діям або агентам, а ребра визначають переходи між ними (рис. 2.4). Саме такий підхід використовує LangGraph: у документації зазначено, що агентні workflow моделюються як графи, де поведінка визначається через вузли, стан і переходи. LangGraph також робить акцент на stateful workflows, durable execution, streaming і human-in-the-loop. Цей тип архітектури зручний для складних агентів, де можливі різні маршрути виконання: перевірка, повторення, повернення до попереднього кроку, втручання людини.

Рисунок 2.3 – Workflow-архітектура

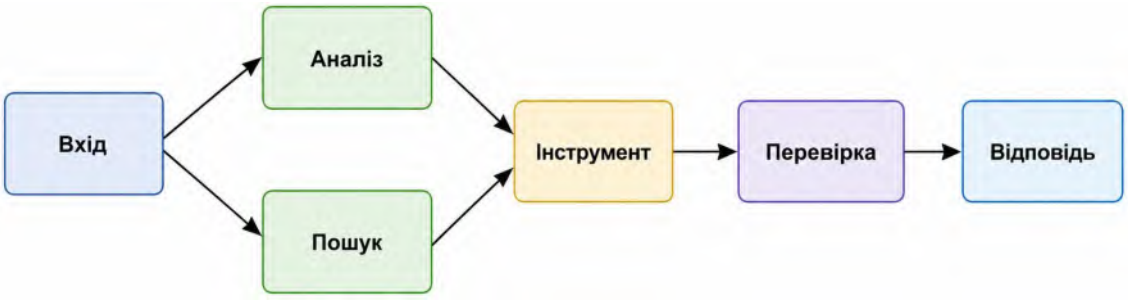

Рисунок 2.4 – Graph-архітектура

5. Multi-agent архітектура – система складається не з одного агента, а з кількох спеціалізованих агентів (рис. 2.5). Кожен агент має свою роль.

Наприклад, один агент шукає інформацію, другий аналізує, третій перевіряє, четвертий формує фінальний текст. AutoGen описує агентів як самостійні модулі, які можна розробляти, тестувати, повторно використовувати й об’єднувати у складніші multi-agent системи. Ця архітектура підходить для складних задач, де корисно розділити відповідальність: coding agent, data agent, research agent, reviewer agent тощо [5].

Рисунок 2.5 – Multi-agent архітектура

6. Role-based / Crew-архітектура. Це різновид multi-agent підходу, де агенти організовані як «команда» з ролями, цілями та задачами (рис. 2.6). Наприклад, у CrewAI агент визначається як автономна одиниця, яка може виконувати задачі, приймати рішення відповідно до ролі й мети, використовувати інструменти, співпрацювати з іншими агентами та підтримувати пам’ять. Поняття «crew» у CrewAI означає групу агентів, що працюють разом для виконання набору задач. Ця архітектура добре підходить для імітації командної роботи: дослідження, написання звітів, аналізу документів, генерації контенту [28].

7. Supervisor / Router архітектура – головний агент-диспетчер, який визначає, кому передати задачу (рис. 2.7). Наприклад: питання про оплату → фінансовий агент; питання про технічну помилку → технічний агент; питання про документи → агент для пошуку файлів. Схожий принцип

використовується у handoff-архітектурах: OpenAI Agents SDK описує handoffs як механізм, за допомогою якого агент може делегувати задачу іншому спеціалізованому агенту [29].

8. Event-driven / Flow архітектура (рис. 2.8). У цьому випадку робота агента запускається подіями: новий лист, новий файл, повідомлення в чаті, оновлення таблиці, натискання кнопки. CrewAI Flows описуються як структуровані event-driven workflows, які дозволяють поєднувати задачі, керувати станом і контролювати перебіг виконання AI-додатків [6]. Ця архітектура близька до n8n [30], Make.com [31] та інших платформ автоматизації, де агент є частиною ширшого процесу.

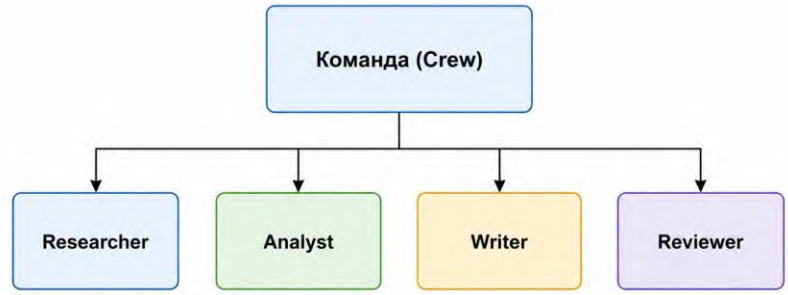

Рисунок 2.6 – Role-based / Crew-архітектура

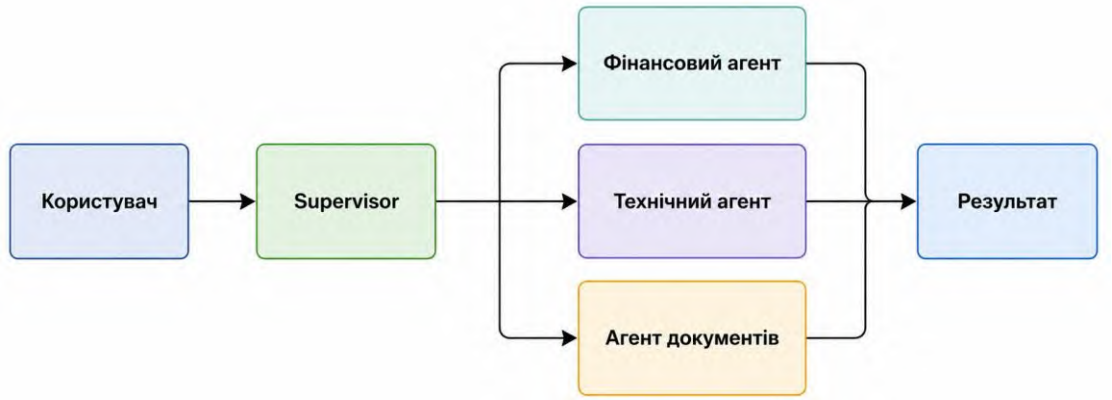

Рисунок 2.7 – Supervisor / Router архітектура

Рисунок 2.8 – Event-driven / Flow архітектура

9. RAG-архітектура агента (рис. 2.9). У RAG-архітектурі агент перед відповіддю звертається до зовнішньої бази знань. Це важливо для агентів, які працюють з документами, інструкціями, технічною документацією, локальними файлами або корпоративними базами знань. Такий агент не покладається лише на знання моделі, а використовує актуальний контекст.

10. Human-in-the-loop архітектура – людина залишається частиною процесу й може підтверджувати або коригувати дії агента (рис. 2.10). Це корисно для ризикованих дій: надсилання листів, видалення файлів, фінансових операцій, зміни даних у CRM. LangGraph окремо виділяє human-in-the-loop як одну з важливих можливостей для агентної оркестрації [4].

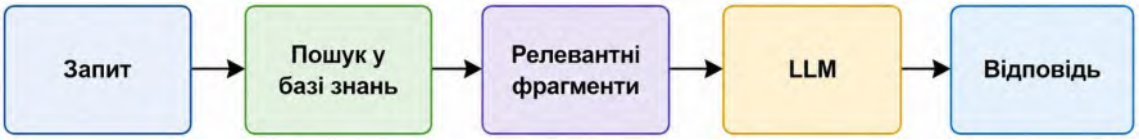

Рисунок 2.9 – RAG-архітектура

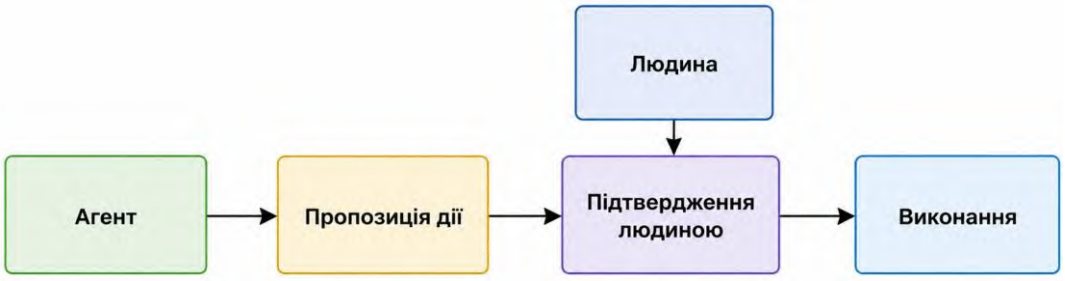

Рисунок 2.10 – Human-in-the-loop архітектура

11. Local / Edge архітектура (рис. 2.11). У локальній архітектурі агент працює без хмарних сервісів або з мінімальним їх використанням. Цей підхід важливий для конфіденційних даних. Наприклад, локальний агент може працювати на ПК, сервері підприємства або Raspberry Pi [32], використовувати LM Studio [33], Ollama [34], KoboldCpp [35], локальну векторну базу та локальні інструменти. локальна архітектура AI-агента забезпечує обробку даних у межах локальної інфраструктури без передавання конфіденційної інформації до зовнішніх хмарних сервісів.

Рисунок 2.11 – Local / Edge архітектура

На основі проведених досліджень сформована узагальнена узагальнена класифікація типових архітектур фреймворків AI-агентів (табл. 2.1).

Таблиця 2.1 – Типові архітектур фреймворків AI-агентів

Архітектура	Суть	Приклади застосування
Одноагентна	Один агент виконує задачу	Чат-бот, асистент
Tool-calling / ReAct	Агент викликає інструменти	Пошук, API, файли
Workflow	Жорстко задана послідовність дій	Автоматизація процесів
Graph	Логіка подана як граф станів	Складні Agent Workflows
Multi-agent	Кілька агентів співпрацюють	Аналіз, розробка, дослідження
Crew / Role-based	Команда агентів із ролями	Researcher, writer, reviewer
Supervisor / Router	Головний агент розподіляє задачі	Підтримка користувачів
Event-driven	Агент запускається подією	Лист, файл, повідомлення
RAG-based	Агент працює з базою знань	Документи, інструкції
Human-in-the-loop	Людина підтверджує дії	Критичні операції
Local / Edge	Робота без хмари	Конфіденційні дані

Зазвичай, сучасні фреймворки AI-агентів поєднують кілька архітектур одночасно. Наприклад, локальний агент може бути RAG-based, використовувати tool-calling, мати graph-логіку та працювати у multi-agent режимі. Але в цілому, можна виділити 3 основні групи: одноагентні архітектури, workflow/graph-архітектури та multi-agent архітектури.

2.4 Реалізація локального AI-агента

Окремої уваги заслуговує розвиток локальних AI-агентів, оскільки саме цей напрям сьогодні є одним із найбільш перспективних для практичного впровадження систем AI в організаціях, де важливими є конфіденційність,

автономність і контроль над даними. Під локальним AI-агентом доцільно розуміти програмну систему, яка використовує мовну модель або інші AI-компоненти без обов’язкового звернення до хмарних сервісів. Такий агент може працювати на локальному комп’ютері, сервері організації або в закритій внутрішній мережі, виконуючи завдання користувача за допомогою підключених інструментів, баз знань, файлів, баз даних або інших програмних модулів.

З поширенням локальних великих мовних моделей, малих мовних моделей та спеціалізованих платформ для їх запуску стало можливим створення агентних систем, які не потребують постійного підключення до зовнішніх AI-сервісів. Якщо раніше більшість інтелектуальних систем залежали від комерційних API, то сьогодні дедалі частіше застосовуються локальні рішення, які дозволяють запускати моделі безпосередньо на власному обладнанні. Це може бути персональний комп’ютер, локальний сервер, мінікомп’ютер або навіть одноплатний пристрій, залежно від складності задачі та вимог до продуктивності.

Основою локального AI-агента є мовна модель, яка відповідає за розуміння запиту користувача, формування відповіді та прийняття рішень на подальші дії. Однак сам агент не обмежується лише генерацією тексту. Його особливість полягає в тому, що він може взаємодіяти з іншими компонентами системи: читати документи, виконувати пошук у локальній базі знань, запускати скрипти, обробляти файли, аналізувати дані або передавати результати в інші програмні модулі. Таким чином, локальний AI-агент виступає не просто як чат-бот, а як інтелектуальний посередник між користувачем, LLM і інформаційною інфраструктурою організації. Важливою перевагою локальних AI-агентів є підвищений рівень захисту інформації. Оскільки обробка даних відбувається всередині локального середовища, конфіденційні документи, звернення користувачів, персональні дані або службова інформація не передаються стороннім сервісам. Це особливо актуально для підприємств, медичних установ, навчальних закладів,

державних організацій та компаній, які працюють із внутрішньою документацією або комерційною таємницею. Локальне розгортання дозволяє краще контролювати доступ до інформації, обмежувати коло користувачів системи та зменшувати ризики витоку даних.

Ще однією перевагою локальних агентів є незалежність від зовнішніх постачальників AI-сервісів. Використання хмарних моделей часто передбачає оплату за кількість запитів, обмеження за швидкістю роботи, залежність від доступності сервера постачальника та необхідність дотримання його правил використання. У випадку локального агента організація отримує більшу автономність: система може працювати без постійного доступу до інтернету, її можна адаптувати під конкретні задачі, а витрати після розгортання стають більш прогнозованими. Це робить локальні AI-агенти доцільними для довготривалого використання в закритих або напівзакритих інформаційних системах. Локальні AI-агенти можуть застосовуватися для різних практичних завдань. Наприклад, вони можуть допомагати в обробці звернень користувачів, аналізі текстових документів, пошуку відповідей у внутрішній базі знань, підготовці звітів, автоматизації рутинних операцій або попередньому аналізі великих обсягів інформації. У поєднанні з технологією RAG локальний агент може не лише відповідати на загальні запитання, а й використовувати конкретні документи організації як джерело знань. Це значно підвищує практичну цінність системи, оскільки відповіді формуються з урахуванням актуальної локальної інформації.

Разом із перевагами локальні AI-агенти мають і певні обмеження. Насамперед їх ефективність залежить від продуктивності обладнання, на якому запускається модель. Потужні мовні моделі потребують значного обсягу ОЗП, швидкого процесора або графічного прискорювача. Тому під час проєктування локальної системи необхідно враховувати баланс між якістю роботи моделі, швидкістю відповіді та доступними технічними ресурсами. Крім того, локальні моделі можуть поступатися найсучаснішим хмарним рішенням за якістю генерації, однак для багатьох прикладних задач цього

рівня достатньо, особливо якщо система працює з вузькою предметною областю.

Під час реалізації локального AI-агента важливо обрати такий фреймворк, який не перевантажує систему зайвими залежностями, може працювати з локальними або OpenAI-сумісними мовними моделями та дозволяє гнучко описувати логіку роботи агента. У цьому контексті доцільно розглянути три легковагові рішення: PicoClaw [36], PicoFlow [37] та PocketFlow [38]. Їх можна віднести до класу мінімалістичних інструментів для створення AI-агентів, які протиставляються великим фреймворкам із великою кількістю абстракцій і залежностей. PicoClaw можна розглядати як готове рішення для створення персонального AI-асистента. Проєкт позиціонується як ультралегкий AI Assistant, написаний мовою Go, із фокусом на запуск на малопотужному обладнанні. У документації зазначено, що PicoClaw орієнтований на невелике споживання оперативної пам’яті, швидкий старт і можливість розгортання на дешевому або edge-обладнанні. Також у PicoClaw передбачено WebUI, gateway-режим, роботу з різними LLM-провайдерами та підтримку локального розгортання через Ollama або VLLM. Це робить його придатним для побудови локального агента, який може виконувати роль персонального помічника, обробляти запити користувача, запускати інструменти та працювати з локальною мовною моделлю без обов’язкового використання хмарних сервісів. Водночас розробники PicoClaw прямо зазначають, що проєкт перебуває на ранньому етапі розвитку і не рекомендується для production-використання до версії 1.0. PicoFlow має дещо інше призначення. Якщо PicoClaw більше схожий на готового асистента, то PicoFlow є невеликим Python-фреймворком для опису агентних workflow. Його основна ідея – логіка агента описується як послідовність явних кроків: кожен flow є окремою функцією, а оператор >> використовується для об’єднання кроків у зрозумілий pipeline. PicoFlow підтримує послідовне виконання, цикли, паралельні гілки, об’єднання результатів, роботу з інструментами, трасування, таймаути та потокову

генерацію відповіді. Окремою перевагою є підтримка OpenAI-сумісних endpoint-ів, зокрема можливість підключення локальної моделі через адресу на кшталт 127.0.0.1:8000 або Ollama. Тому PicoFlow зручно використовувати тоді, коли потрібно не просто запустити чат-асистента, а побудувати контрольований сценарій: наприклад, попередня обробка запиту, класифікація, звернення до локальної LLM, виклик інструменту, перевірка відповіді та формування фінального результату. PocketFlow також належить до мінімалістичних фреймворків, але його концепція базується на графовій моделі. У документації PocketFlow зазначено, що ядро фреймворку реалізує базову абстракцію Graph + Shared Store: окремі вузли виконують задачі, flow з'єднує ці вузли, а спільне сховище використовується для обміну даними між частинами workflow. PocketFlow підтримує такі шаблони, як agents, workflow, RAG, map-reduce, structured output і multi-agent-сценарії. Важлива особливість PocketFlow полягає в тому, що він не нав'язує конкретні вбудовані утиліти для LLM, векторних баз чи пошуку, а пропонує реалізовувати їх окремо. Це зменшує залежність від конкретного постачальника моделей і робить фреймворк зручним для навчальних та дослідницьких робіт, де потрібно показати сам принцип побудови агента.

Окрім розглянутих спеціалізованих фреймворків для реалізації локального AI-агента доцільно розглянути платформу n8n [30]. Згідно [39], один з варіантів наведений на рис. 2.12.

На відміну від мінімалістичних Python- або Go-фреймворків, n8n орієнтований не лише на програмну реалізацію агентної логіки, а й на візуальне проєктування автоматизованих сценаріїв. У n8n робочий процес будується як послідовність вузлів, кожен із яких відповідає за окрему дію: отримання вхідних даних, виклик мовної моделі, обробку відповіді, звернення до зовнішнього сервісу, збереження результату або передавання його користувачу. Наприклад, у [40] описано підхід до створення локального AI-агента за допомогою трьох основних компонентів: Ollama [34], n8n та Docker [41]. Ollama використовується для запуску LLM локально, n8n – для

побудови самого AI-агента, а Docker – для простого запуску всіх застосунків на комп'ютері користувача. Важливо, що така система може працювати локально, без оплати API-токенів і без обов'язкового використання хмарних сервісів.

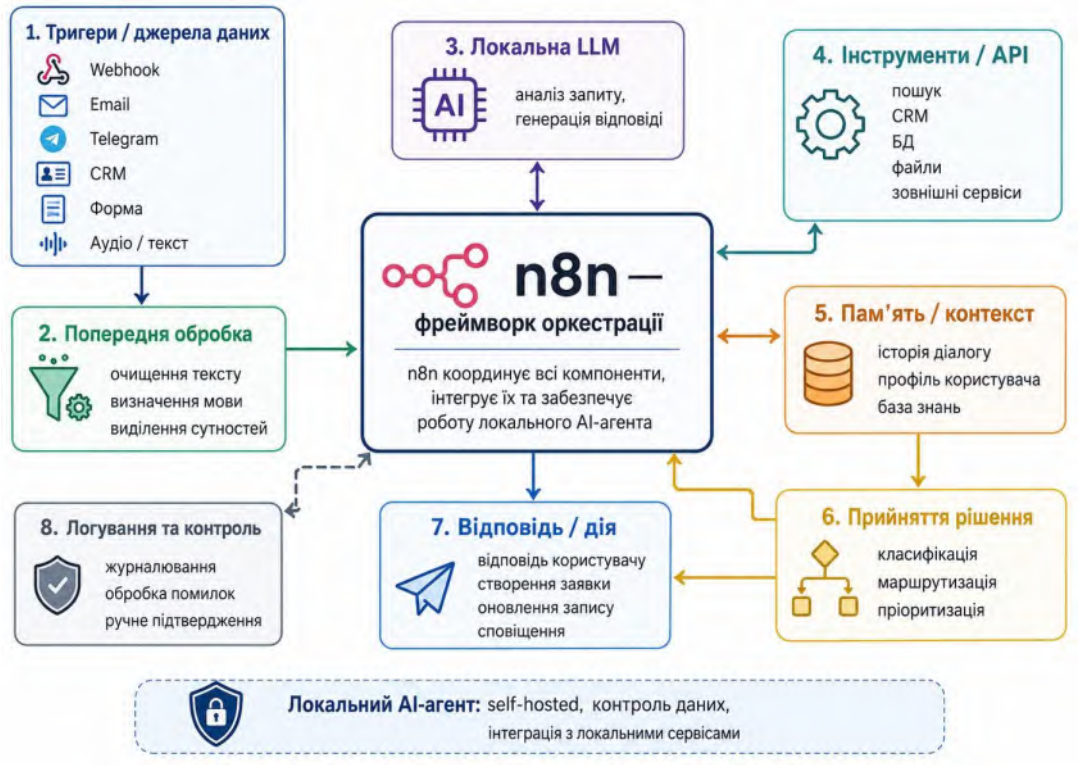

Рисунок 2.12 – Типова структура AI-агента на базі n8n

Особливо важливою є наявність у n8n вузла «AI Agent». Згідно з документацією n8n, AI Agent node призначений для створення автономної системи, яка отримує дані, приймає рішення та може використовувати зовнішні інструменти або API для виконання поставлених завдань. Такий агент може визначати, який інструмент потрібно застосувати залежно від запиту користувача. Це дозволяє використовувати n8n не лише як звичайну систему автоматизації, а саме як платформу для побудови агентних сценаріїв. Локально встановлена n8n має окремий вузол «Ollama Chat Model», який дозволяє підключати локальні LLM до агентних workflow. В документації Ollama для n8n зазначено, що при локальному запуску базова адреса може мати вигляд http://localhost:11434, а якщо n8n працює всередині Docker-

контейнера, може використовуватися адреса http://host.docker.internal:11434. Це важливий практичний момент, оскільки під час контейнеризації localhost всередині контейнера не завжди вказує на основну ОС. З практичної точки зору локальний AI-агент на базі n8n може мати вигляд – рис. 2.13.

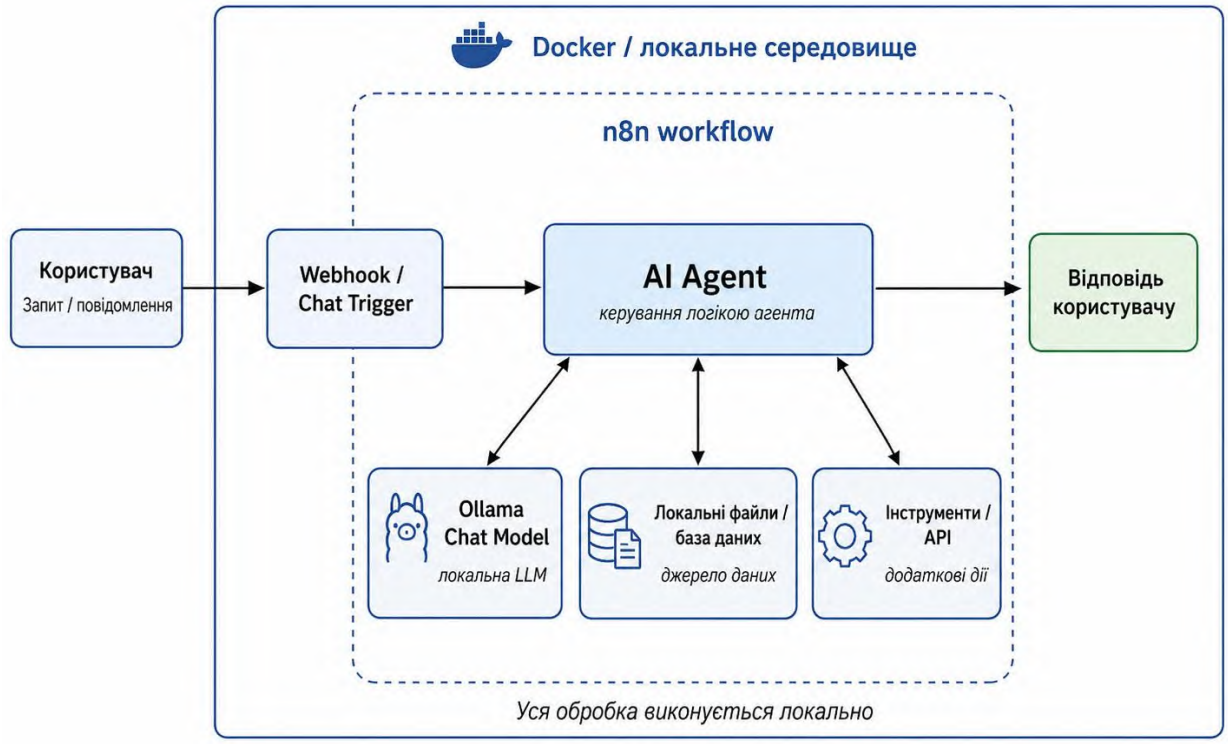

Рисунок 2.13 – Варіант архітектури AI-агента на базі n8n

Таким чином, n8n можна розглядати як low-code платформу для синтезу AI-агента, оскільки вона дозволяє створювати логіку агента не лише шляхом написання програмного коду, а й через графічний редактор. Використання Docker у такій архітектурі є доцільним, оскільки n8n офіційно рекомендує Docker для більшості сценаріїв self-hosted розгортання. У документації зазначено, що Docker забезпечує чисте ізольоване середовище, зменшує проблеми несумісності операційної системи та спрощує керування базами даних і змінними середовища. Тому Docker можна використати для запуску n8n, Ollama, векторної бази даних та інших компонентів локального AI-агента. У порівнянні з PicoClaw, PicoFlow та PocketFlow, платформа n8n має інше призначення (табл. 2.2).

Таблиця 2.2 – Порівняння властивостей фреймворків для синтезу локального AI-агент

Критерій	PicoClaw	PicoFlow	PocketFlow	n8n
Тип рішення	Легковаговий AI-асистент	Python DSL для workflow	Графовий фреймворк	Low-code платформа автоматизації
Основний підхід	Готовий асистент	Програмний pipeline	Граф вузлів і shared store	Візуальний workflow
Локальні LLM	Ollama, vLLM	OpenAI-сумісні endpoint-и, Ollama	Через власну інтеграцію	Ollama Chat Model node
Зручність для студента	Середня	Середня	Висока для пояснення архітектури	Висока для демонстрації роботи
Гнучкість коду	Середня	Висока	Висока	Середня
Перевага	Готовий локальний асистент	Чітка логіка сценаріїв	Простота архітектурної моделі	Наочна побудова агента без складного коду
Недолік	Ранній етап розвитку	Потрібно писати код	Багато компонентів реалізуються вручну	Обмеження в порівнянні з повним програмуванням
Доцільність у роботі	Приклад готового агента	Реалізація власного сценарію	Пояснення принципів агентної архітектури	Практична демонстрація локального AI-агента

Якщо PicoFlow і PocketFlow більше орієнтовані на програмну реалізацію агентної логіки, то n8n дає змогу будувати агента візуально. Це робить його зручним для демонстрації роботи AI-агента (архітектура стає зрозумілою навіть без детального аналізу програмного коду). Водночас n8n менш гнучкий для низькорівневого програмування, ніж Python-фреймворки, але значно зручніший для інтеграції різних сервісів, API, БД і локальних LLM. Платформа n8n розглядається як практичний інструмент для побудови agentic AI з локальною LLM. Його перевага полягає в тому, що він поєднує візуальне проєктування workflow, підтримку AI Agent node, інтеграцію з Ollama та можливість локального розгортання через Docker.

2.5 Застосування Docker для локального AI-агента

Для практичної реалізації локального AI-агента доцільно використати Docker, оскільки контейнеризація дозволяє запускати додаток разом з потрібними залежностями в ізольованому та відтворюваному середовищі (рис. 2.14).

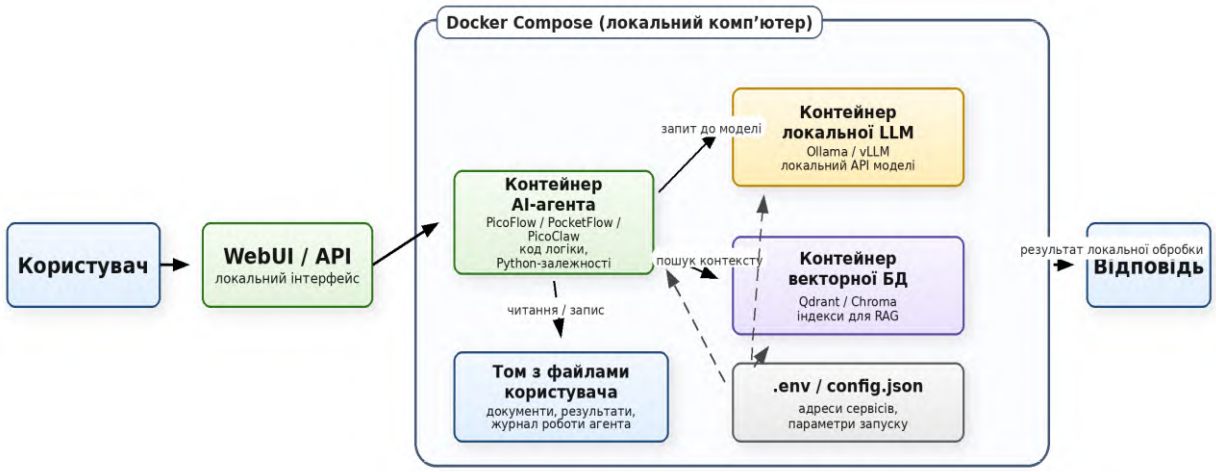

Рисунок 2.14 – Реалізація AI-агента з використанням Docker

У випадку AI-агента це особливо важливо, оскільки система може містити кілька компонентів: сам агентний фреймворк, локальну LLM, векторну БД, сховище файлів, WebUI або API-сервер. Docker Compose дає змогу описати такі компоненти в одному compose.yml-файлі, визначити змінні середовища, томи для збереження даних і внутрішню мережу між контейнерами.

Відповідно, у роботі запропонована така структуру локального розгортання: контейнер AI-агента – містить PicoFlow або PocketFlow, Python-залежності та код логіки агента; контейнер локальної LLM (наприклад, Ollama або vLLM, який обслуговує мовну модель через локальний API); контейнер векторної БД (наприклад, Qdrant, Chroma або інше сховище для RAG-сценаріїв); том для файлів користувача (окрема директорія, у якій зберігаються документи, проміжні результати та журнал роботи агента); файл

.env (для зберігання налаштувань, адрес локальних сервісів і параметрів запуску). Для PicoClaw Docker вже існує сценарій запуску через docker compose, з генерацією конфігурації, редагуванням config.json і запуском launcher-режиму (рис. 2.15).

```bash
# 1. Клонування репозиторію
git clone https://github.com/sipeed/picoclaw.git
cd picoclaw
# 2. Перший запуск — автоматично створює docker/data/config.json і завершується
docker compose -f docker/docker-compose.yml --profile launcher up
# 3. Редагування конфігурації
vim docker/data/config.json
# 4. Запуск launcher-режиму
docker compose -f docker/docker-compose.yml --profile launcher up -d
# 5. Відкрити WebUI
http://localhost:18800
```

Рисунок 2.15 – Сценарій запуску PicoClaw через docker compose

Перший запуск контейнера з профілем launcher автоматично створює файл конфігурації docker/data/config.json, після чого користувач редагує параметри моделі, ключі доступу та інші налаштування. Подальший запуск виконується командою docker compose -f docker/docker-compose.yml --profile launcher up -d, після чого вебінтерфейс PicoClaw доступний локально за адресою localhost:18800. Такий підхід дає змогу швидко розгорнути AI-агента без ручного встановлення залежностей у системне середовище користувача.

Це дозволяє швидко розгорнути локального AI-агента без ручного встановлення всіх залежностей у систему користувача. Тобто, Docker використовується як інфраструктурний засіб, що спрощує розгортання, перенесення та тестування локального AI-агента. Його застосування дає змогу масштабування рішення: той самий AI-агент може бути запущений на різних комп’ютерах за однаковим сценарієм.

РОЗДІЛ 3
РЕКОМЕНДАЦІЇ ЩОДО ПРАКТИЧНОЇ РЕАЛІЗАЦІЇ AI-АГЕНТА

3.1 Експериментальна реалізація AI-агента

У межах експериментальної реалізації запропонований AI-агент. Основна ідея реалізації полягає в тому, що користувач надсилає текстове повідомлення у Telegram-чат (рис. 3.1), після чого PicoClaw приймає це повідомлення через Telegram-канал, передає його до LLM, що запущена у середовищі LM Studio (рис. 3.2), отримує згенеровану відповідь і повертає її назад користувачу у той самий чат (Додаток А).

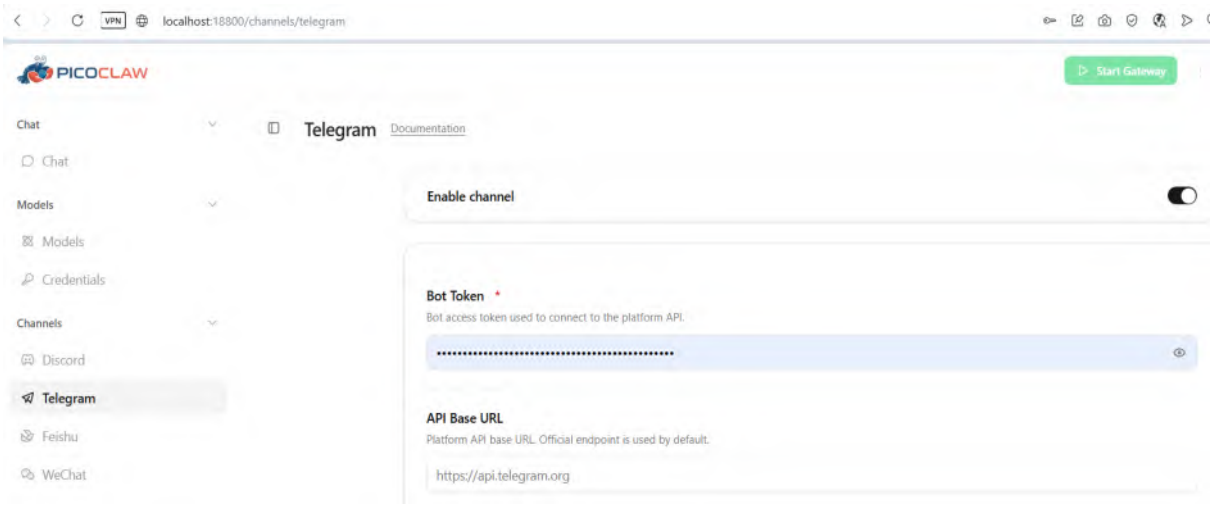

Рисунок 3.1 – Підключення до PicoClaw Telegram-каналу

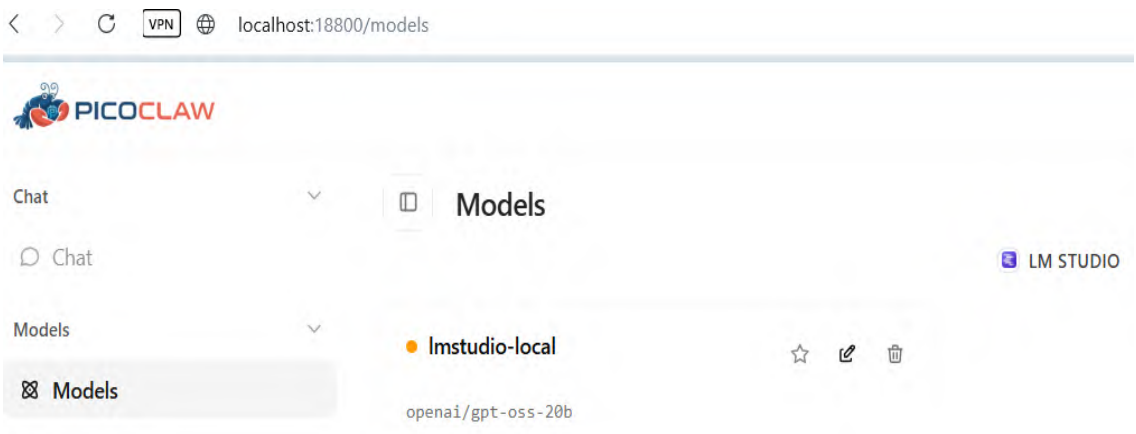

Рисунок 3.2 – Підключення до PicoClaw середовища LM Studio

Такий підхід дозволяє продемонструвати практичне застосування фреймворку PicoClaw для синтезу AI-агента, який працює не лише як окрема консольна програма, а як повноцінний чат-асистент, інтегрований з популярним месенджером. У запропонованій архітектурі Telegram виконує роль користувацького інтерфейсу, через який здійснюється введення запитів та отримання відповідей.

Згідно з документацією PicoClaw, Telegram є рекомендованим каналом взаємодії, оскільки він достатньо простий у налаштуванні та підтримує не лише текстові повідомлення, а й додаткові можливості, зокрема роботу з голосовими повідомленнями за наявності відповідної ASR-моделі. Для створення Telegram-бота використовується стандартний механізм BotFather: створюється новий бот, отримується токен доступу, після чого цей токен додається до конфігурації PicoClaw. Також у конфігурації може бути вказаний параметр allow_from, який обмежує доступ до асистента лише певними користувачами Telegram, що є важливим для тестування та захисту системи від стороннього використання. Після налаштування каналу запуск взаємодії здійснюється командою picoclaw gateway, яка переводить PicoClaw у режим очікування повідомлень із підключених чат-каналів.

Центральним елементом експериментальної системи є PicoClaw, який виконує роль проміжного програмного шару між Telegram і мовною моделлю. Він приймає повідомлення з Telegram, формує запит до AI-агента, звертається до вибраного провайдера мовної моделі та повертає результат у чат. У цьому випадку як провайдер використовується LM Studio API. Відповідно до документації PicoClaw, LM Studio розглядається як локальний менеджер LLM-моделей, що запускає моделі на комп’ютері користувача та надає OpenAI-сумісний API. Це означає, що PicoClaw може взаємодіяти з локальною моделлю приблизно так само, як із хмарними API, але фактична генерація відповіді виконується локально, без передавання даних до зовнішнього сервісу. Такий підхід є доцільним для бакалаврської роботи, оскільки дозволяє показати практичне застосування локального AI-агента з

підвищеним рівнем контролю над даними. Для роботи з LM Studio спочатку запускається локальний сервер у розділі Developer → Local Server, після чого завантажується вибрана мовна модель. Документація PicoClaw зазначає, що типовою адресою локального API є http://localhost:1234/v1, а ідентифікатор моделі з LM Studio використовується в PicoClaw із префіксом lmstudio/. Наприклад, якщо в LM Studio завантажено певну LLM-модель, її ідентифікатор додається до конфігурації PicoClaw як модель типу lmstudio/your-model-id. У разі локального запуску API-ключ за замовчуванням не потрібен, оскільки LM Studio орієнтована на локальне використання, однак за потреби автентифікація може бути додатково ввімкнена.

Логіка роботи реалізованого Telegram AI-асистента має послідовний характер. Спочатку користувач відкриває Telegram-бота та надсилає йому текстовий запит, наприклад запитання або коротке завдання. Далі Telegram передає це повідомлення до PicoClaw, який працює в режимі gateway. PicoClaw визначає, що повідомлення надійшло з Telegram-каналу, перевіряє дозвіл користувача відповідно до налаштувань доступу та передає текст у модуль AI-агента. Після цього агент звертається до локального API LM Studio, де вже завантажена мовна модель. Модель обробляє запит, генерує відповідь природною мовою та повертає її через OpenAI-сумісний інтерфейс назад до PicoClaw. Завершальним етапом є надсилання відповіді у Telegram-чат, де користувач бачить результат роботи асистента.

Важливою перевагою такої реалізації є розділення функцій між компонентами системи. Telegram відповідає лише за зручний інтерфейс спілкування, PicoClaw – за маршрутизацію повідомлень, логіку роботи агента та інтеграцію з каналами, а LM Studio – за локальне завантаження та виконання мовної моделі. Завдяки цьому система є гнучкою: за потреби можна змінити модель у LM Studio, не змінюючи сам Telegram-бот, або підключити інший канал взаємодії без повної перебудови логіки AI-агента. Крім того, документація PicoClaw вказує, що швидкість роботи локальної моделі залежить від характеристик CPU/GPU, розміру моделі та параметрів

паралельної обробки, тому під час експериментів необхідно враховувати апаратні обмеження комп'ютера.

Таким чином, експериментальна реалізація Telegram AI-асистента на основі PicoClaw та LM Studio демонструє практичний приклад застосування фреймворків для синтезу AI-агента. У такій системі користувач взаємодіє з агентом у звичному середовищі Telegram, PicoClaw забезпечує приймання та обробку повідомлень, а LM Studio виконує локальну генерацію відповідей за допомогою LLM-моделі. Отримана архітектура може бути використана як базовий приклад локального AI-асистента, який не потребує обов'язкового використання хмарного LLM-провайдера та може бути адаптований для навчальних, консультаційних або інформаційно-довідкових задач.

3.2 Формування структурованих інструкцій для AI-агентів

AI-агент працює на основі структурованих інструкцій, які спрямовують систему на досягнення мети. Вони схожі на prompts, що визначають основне завдання агента та правила його поведінки. Розгляньмо ключові особливості AI-агентів, які можуть бути задані такими інструкціями (рис. 3.3).

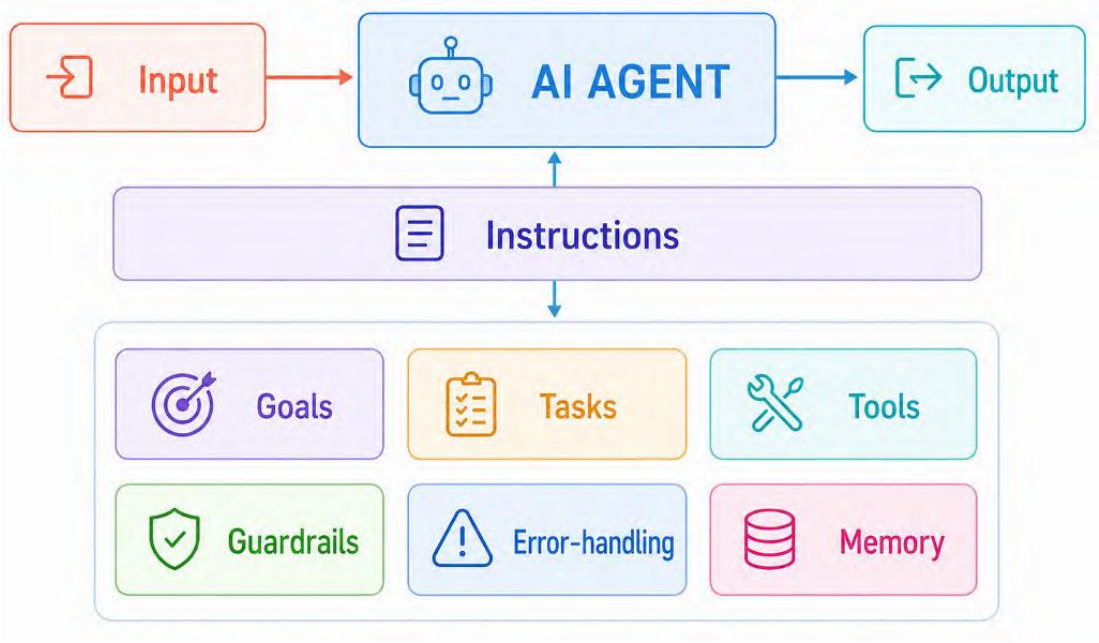

Рисунок 3.3 – Інструкції для AI-агентів

1. Цілі задають високорівневе завдання AI-агента та визначають, хто він такий. Щоб направити AI-агента у процесі виконання, далі йдуть інструкції до завдання – вони пояснюють, що саме повинен робити агент і як він це повинен робити. На рис. 3.4 наведений простий приклад prompt, де вказано, хто такий агент («Search agent»), що він повинен робити («Допомагати користувачу шукати інформацію в Інтернет та зберігати результати на запит») і як він може це виконувати (за допомогою конкретного інструменту).

```
search_agent = Agent
(
        name="Search agent",
        instructions="Help the user search the internet and save results if asked.",
        tools=[WebSearchTool(),save_results],
)
```

Рисунок 3.4 – Спрощений приклад інструкції пошукового AI-агента

2. Інструменти – це зовнішні функції та ресурси, що розширюють можливості AI-агента. В інструкції необхідно описувати, якими інструментами агент може оперувати, наприклад, векторні БД для семантичного пошуку або API для отримання актуальних даних. Агент самостійно вибирає, який інструмент використовувати на кожному етапі, виходячи з характеру завдання, при цьому діючи в рамках заздалегідь заданих обмежень (рамок) – (Guardrails).

3. AI guardrails – це обмеження та політики безпеки, які забезпечують роботу агента в межах етичних та функціональних рамок [42]. Наприклад, можна задати інструкцію підтримувати єдиний tone of voice бренду, щоб агент не скочувався в невідповідний стиль спілкування. Один Guardrail – це явно недостатньо. Треба сприймати їх як багаторівневу систему захисту, де кожен Guardrail відповідає за певний клас потенційних ризиків. У prompt має бути включена логіка обробки помилок: як виявляти збої, коли та як повторювати операції, та у яких випадках необхідно передати проблему на

вищий рівень обробки, коли система або AI-агент не може самостійно її вирішити.

4. Пам’ять (Memory) – це сховище попередніх взаємодій, фактів та результатів використання інструментів, до якого агент може звертатися під час своєї роботи. Користувач може вказати точні інструкції, яку інформацію зберігати та в які моменти. Це можуть бути переваги користувача, результати роботи інструментів, витягнуті сутності (наприклад, дати або локації), важливі віхи та ін. Також можна керувати тим, як AI-агент стискає та чистить пам'ять, щоб уникнути перевантаження: наприклад, шляхом сумаризації минулих взаємодій, видалення застарілих даних або пріорітизації критично важливої інформації. Наприклад, AI-агент у сфері туризму, що допомагає планувати поїздки, може бути проінструктований видаляти застарілі маршрути після завершення подорожі, зберігаючи ключові переваги: «Вважає за краще 4-зіркові готелі; уникає стикувань; веганське харчування». Розглянемо процес бронювання готелю як приклад, щоб продемонструвати, як функціонують AI-агенти та які елементи інфраструктури дозволяють їм розуміти, обробляти та діяти на основі інформації (рис. 3.5).

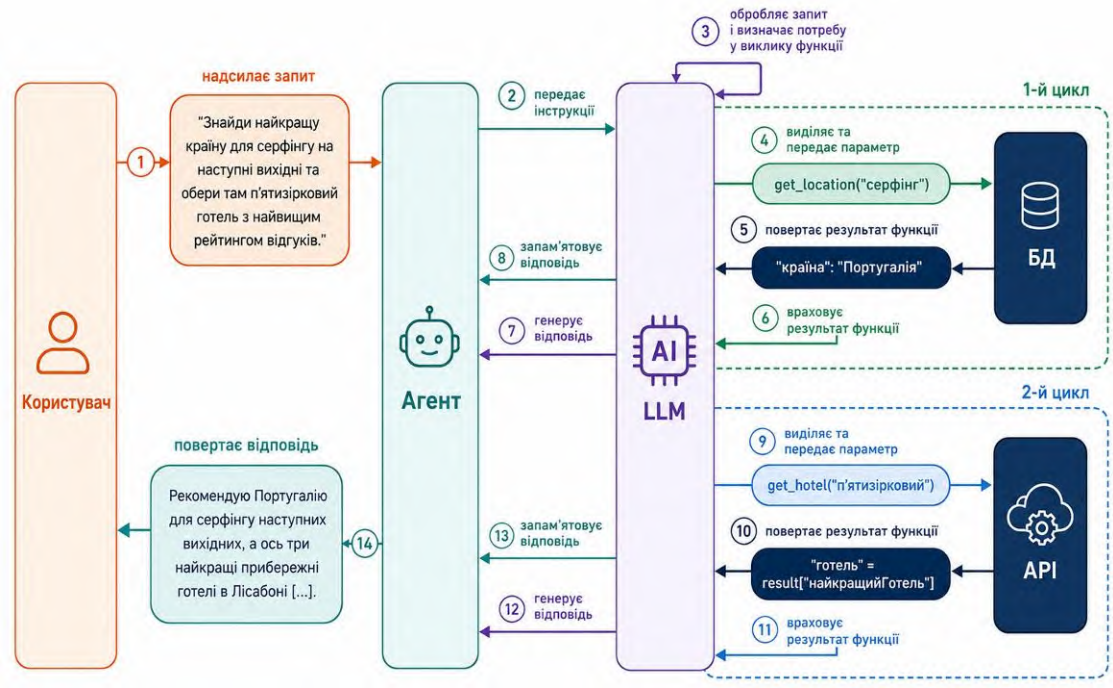

Рисунок 3.5 – Пояснення як працюють AI-агенти

Сприйняття (Perception). Процес починається з того, що користувач надсилає запит через текст або голосове введення. Наприклад: «Знайди для мене найкращу країну для серфінгу на наступні вихідні та вибери п'ятизірковий готель із найвищим рейтингом». AI-агент приймає цей prompt і передає його до LLM.

Міркування та дія (Reasoning and Action) – агент не обробляє складний запит одномоментно –він циклічно проходить через схему: введення інформації → прийняття рішення → дія → фіксація результату → повтор. Кожен такий цикл – це окремий крок, що наближає до фіналу. У даному випадку, LLM аналізує запит, розбиває його на два підзавдання та виділяє ключові параметри – «кращий напрямок для серфінгу» та «5-зірковий готель». Далі модель визначає, що обидва підзавдання вимагають function calling (LLM не має повної та актуальної інформації щодо вузьких предметних областей). Проблема вирішується за допомогою RAG, при якому модель отримує доступ до спеціалізованих зовнішніх джерел даних. Для цього використовується векторна БД (рис. 3.6), в якій кожен документ представлений у вигляді числового ембедингу (точки у багатовимірному просторі, що відображає характеристики тексту).

Рисунок 3.6 – Векторна БД

Коли надходить запит користувача, він також перетворюється на embedding, і система швидко знаходить найближчі збіги у векторному

сховищі. В рамках поточного сценарію система звертається до векторної бази, щоб отримати найбільш релевантні документи про умови для серфінгу з огляду на такі параметри, як висота хвиль та напрям вітру. Припустимо, система визначає, що Португалія – оптимальний напрямок. Цей результат зберігається у часовій (short-term) пам'яті AI-агента. Завдяки цьому надалі не потрібно повторно звертатися до інструменту – модель просто витягує збережене значення з пам'яті. Потім виконується ще один цикл – для другого підзавдання (бронювання готелю). На цей раз AI-агент використовує інший інструмент – інтегрований hotel API, щоб знайти 5-зіркові готелі у вибраній локації (Португалії). Функція повертає актуальну інформацію про доступні варіанти та ціни, а LLM сортує результати за найвищим рейтингом відгуків.

Повернення результатів. Цикл завершується лише при досягненні чітко визначеної умови виходу. У нашому випадку – коли обидва підзавдання виконані. Після того, як вся необхідна інформація отримана, LLM формує відповідь природною мовою: «Рекомендую Португалію для серфінгу в наступні вихідні. Ось три найкращі готелі на першій лінії в Лісабоні [...]». Може бути ситуація, коли відповідь користувача не влаштовує. Припустимо, що він повідомляє AI-агенту: при виборі кращого місця для серфінгу не було враховано критично важливу змінну – температуру води. Отримавши такий зворотний зв'язок, AI-агент знову запускає описаний вище цикл але тепер з урахуванням додаткового параметра. У результаті користувач приймає нову, уточнену пропозицію: Канарські острови.

3.3 Типи агентів та їх можливості

Проаналізуємо два варіанти: системи з одним AI-агентом (Single-agent) та multi-agent система. У single-agent архітектурі один агент відповідає за виконання всього ланцюжка дій: обробляє користувальницький input; вибирає і викликає потрібні інструменти (API, бази даних та ін.); оновлює пам'ять; повторює цикл, доки завдання не буде вирішено. Якщо додати нові

можливості, наприклад, сьогодні це API для пошуку авіаквитків, а завтра – для бронювання готелів, то користувач просто навчає того ж самого AI-агента, коли і як викликати кожен інструмент. Multi-agent системи, навпаки, ділять робочий процес між кількома спеціалізованими агентами. У manager pattern один центральний агент управляє виконанням завдання та делегує subtasks спеціалізованим допоміжним агентам через виклик інструментів. Цей патерн підходить, коли ви хочете, щоб один агент взаємодіяв з користувачем і координував весь процес.

У децентралізованій архітектурі усі AI-агенти – рівноправні учасники. Вони передають завдання одне одному залежно від своєї спеціалізації (рис. 3.7). Така модель дозволяє кожному агенту брати на себе виконання завдання та, при необхідності, безпосередньо взаємодіяти з користувачем.

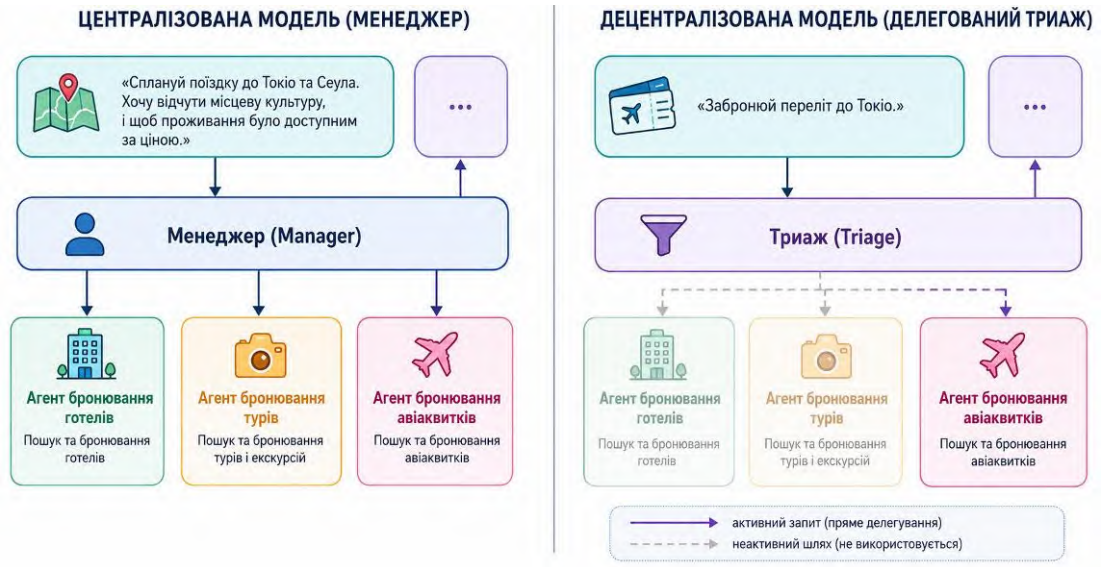

Рисунок 3.7 – Manager pattern та децентралізований підхід

У більшості випадків достатньо одного AI-агента з (ймовірно, постійно зростаючим) набором інструментів. Найкращою практикою вважається інкрементальний підхід – розширювати можливості AI-агента поступово: додавати нові інструменти та уточнювати інструкції, доки він справляється з завданнями. Тільки коли проект стає настільки складним (наприклад, агент починає плутатися в логіці, неправильно інтерпретує інструкції або постійно

вибирає не інструменти), тоді варто розглянути масштабування системи за рахунок впровадження додаткових спеціалізованих AI-агентів. Не всі агентні системи працюють однаково. Залежно від характеру завдання працювати включаються різні типи агентів (рис. 3.8).

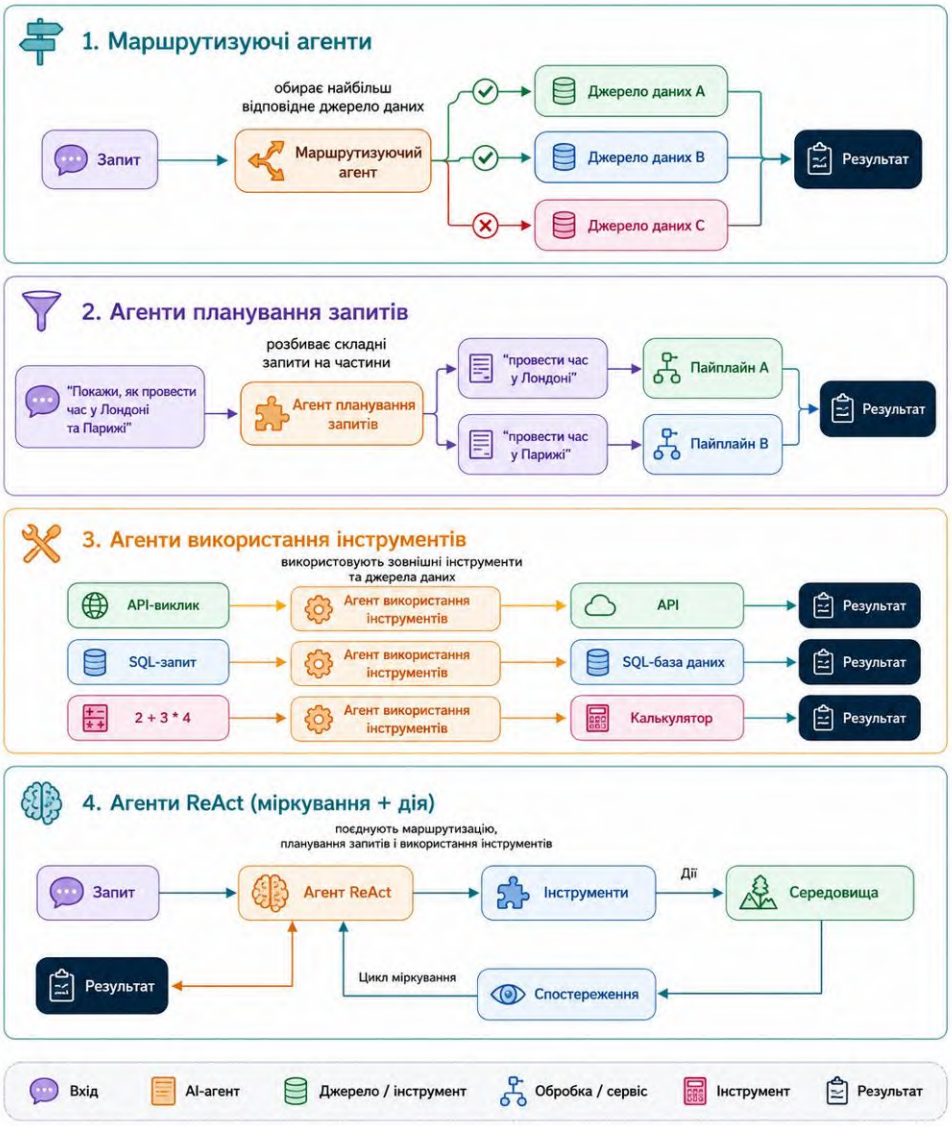

Рисунок 3.8 – Типи AI-агентів

Кожен їх може функціонувати автономно або як частина багатоагентної архітектури.

Routing-агенти аналізують запит користувача, оцінюють його складність і вибирають відповідне джерело даних. Їхнє завдання – правильно маршрутизувати запит залежно від змісту та необхідного рівня обробки.

Query planning-агенти розбивають складні запити на незалежні підзавдання, які можна виконувати паралельно. Наприклад: користувач запитує – «Розкажи, як можна провести час у Лондоні та Парижі»? Агент ділить prompt на дві частини («провести час у Лондоні» та «в Парижі»), відправляє їх у різні pipeline, а потім поєднує результати в єдину відповідь.

Tool use-агенти взаємодіють з зовнішнім функціоналом і вибирають підходящі ресурси – будь то API, БД, калькулятори або навіть інші AI-агентні моделі. Наприклад: запит «Знайди готель у Барселоні дешевше $200» Агент звертається до hotel API, застосовує фільтрацію за ціною та повертає добірку відповідних варіантів.

ReAct-агенти (Reason + Act) поєднують можливості всіх 3-ох типів: маршрутизація, планування запитів та робота з інструментами. Вони вирішують складні завдання, послідовно розбиваючи їх на підзавдання та вирішуючи кожну окремо. Спочатку агент розмірковує про те, як краще підійти до проблеми, потім виконує дію (наприклад, робить запит до БД або викликає інструмент). Після виконання він аналізує результат, робить висновки та коригує наступні кроки. Такий цикл – reason → act → refine – повторюється доти, доки завдання не буде успішно вирішено.

3.4 Рекомендації з вибору інструментарію для реалізації AI-агентів

Під час впровадження AI-агентів у робочі процеси важливо враховувати не лише їхню функціональність, а й питання безпеки, конфіденційності та відповідності нормативним вимогам. Оскільки AI-агент може взаємодіяти з внутрішніми системами, обробляти дані користувачів, формувати відповіді, надсилати повідомлення або виконувати певні дії від імені людини, необхідно заздалегідь визначити межі його повноважень. Одним із базових принципів безпечного використання AI-агентів є принцип найменших привілеїв. Його суть полягає в тому, що агент повинен мати лише ті права доступу, які

необхідні для реалізації конкретного завдання. Якщо AI-агенту достатньо лише аналізувати інформацію, йому доцільно надати доступ тільки для читання. Доступ на зміну, створення або видалення даних повинен надаватися окремо для кожної системи та лише у випадках, коли це справді необхідно. Важливим аспектом є також контроль місця зберігання даних і ведення журналів подій. Організація повинна розуміти, де саме зберігаються дані, які передаються або обробляються AI-агентом. Крім того, необхідно забезпечити ведення журналів аудиту, у яких фіксуються дії агента, звернення до систем, зміни даних та інші важливі події. Це дозволяє підвищити прозорість роботи системи та спростити виявлення можливих помилок або порушень. Для критичних операцій доцільно застосовувати підхід «людина в циклі». Це означає, що певні дії AI-агента мають виконуватися лише після підтвердження користувачем або відповідальною особою. До таких дій можна віднести надсилання зовнішніх повідомлень, публікацію інформації, фінансові операції, зміну важливих записів у базах даних або взаємодію з клієнтами від імені компанії. Такий підхід зменшує ризик небажаних (помилкових) дій з боку автоматизованої системи. Окрему увагу слід приділяти перевірці постачальників програмних рішень, які використовуються для створення або розгортання AI-агентів. Перед інтеграцією зовнішніх сервісів варто оцінити їхню відповідність стандартам безпеки, наявність сертифікацій, таких як SOC 2 або ISO, умови обробки персональних даних, наявність договору про обробку даних, а також історію можливих інцидентів безпеки. Така перевірка дозволяє знизити ризики, пов’язані з витоком інформації, несанкціонованим доступом або невідповідністю вимогам законодавства. Під час впровадження agentic AI-інструментів важливо враховувати не лише їхню функціональність, а й модель ціноутворення. AI-агенти можуть виконувати багато автономних дій: звертатися до мовних моделей, здійснювати веб-запити, запускати сценарії, працювати з документами або інтегруватися з різними сервісами. Кожна з цих дій може впливати на загальну вартість використання системи, тому

перед впровадженням необхідно оцінити можливі витрати. Однією з поширених моделей оплати є оплата за користувача або робоче місце. Такий підхід є зручним для команд, які переважно працюють через графічний інтерфейс платформи та використовують інструмент у межах стандартних робочих процесів. У цьому випадку витрати є більш прогнозованими, оскільки організація сплачує фіксовану суму за певну кількість користувачів.

Іншим варіантом є оплата за задачу або запуск сценарію. Така модель є доцільною тоді, коли використання AI-агента має нерівномірний характер, тобто відбувається періодично або у вигляді коротких інтенсивних навантажень. Наприклад, агент може запускатися лише для обробки нових заявок, генерації звітів або виконання окремих автоматизованих операцій. У такому випадку оплата залежить від фактичної кількості запусків або виконаних завдань. Окрему увагу слід приділяти оплаті за токени, обчислення або інші одиниці вимірювання використання. Така модель часто застосовується під час роботи з великими мовними моделями та API-сервісами. Вартість може суттєво зростати у випадках, коли агент використовує довгі контекстні вікна, обробляє великі документи, багаторазово звертається до зовнішніх джерел або виконує велику кількість веб-запитів. Тому при проєктуванні системи необхідно враховувати не лише кількість користувачів, а й обсяг даних, які передаються до моделі та повертаються у відповіді. На практиці багато платформ використовують гібридну модель ціноутворення, яка поєднує кілька підходів одночасно. Наприклад, сервіс може передбачати фіксовану плату за користувача, додаткову оплату за кількість запусків автоматизацій та окрему тарифікацію API-запитів або використаних токенів. Такий підхід ускладнює попереднє планування бюджету, тому потребує уважного аналізу тарифів і можливих сценаріїв використання. Для зменшення фінансових ризиків доцільно заздалегідь встановлювати бюджетні обмеження та сповіщення про перевищення витрат. Це дозволяє своєчасно виявляти надмірне використання ресурсів, помилки в логіці роботи агента або неконтрольовану кількість API-

запитів. Наприклад, неправильно налаштований агент може багаторазово повторювати одну й ту саму дію, надсилати зайві запити до моделі або обробляти надто великі обсяги даних, що призведе до непередбачуваних витрат. Вибір платформи agentic AI доцільно здійснювати з урахуванням конкретного завдання, джерел даних, способу виконання дій, рівня безпеки та бюджету. Насамперед потрібно чітко визначити, яку саме роботу має виконувати агент: наприклад, підготувати й персоналізувати електронні листи, проаналізувати дані, створити звіт або оновити інформаційну панель.

Також важливо визначити, де розміщені необхідні дані: у Gmail, CRM-системі, Google Drive, Snowflake, Jira або інших сервісах. Від цього залежить, які інтеграції та дозволи потрібні агенту для виконання завдань. Окремо слід враховувати поверхню дії: роботу через браузер, API, файли, електронні таблиці, календарі або програмні репозиторії. Не менш важливими є запобіжні механізми. До них належать обмеження прав на читання й запис, перелік дозволених доменів, денні ліміти на надсилання повідомлень, журналювання дій та можливість перевірки результатів. Це актуально коли агент має доступ до корпоративних або конфіденційних даних. Під час вибору також потрібно враховувати бюджет, оскільки різні платформи можуть мати різні моделі оплати: за кількість токенів, виконаних завдань, користувачів або підключених сервісів. Доцільно починати впровадження з тестового режиму. Спочатку агент може лише планувати та моделювати виконання завдання, після чого користувач перевіряє його дії. Далі можна перейти до режиму читання, і лише після цього поступово надавати право запису або виконання дій в одній контрольованій системі. Безпечне використання AI-агентів потребує чіткого визначення прав доступу, контролю обробки даних, ведення журналу дій, залучення людини до підтвердження критичних операцій та оцінки сторонніх сервісів. Дотримання цих положень підвищує надійність системи, зменшує ризики для організації та забезпечити відповідність вимогам інформаційної безпеки.

3.5 Економічне обґрунтування прийнятих рішень

У даній роботі Telegram використовується як інтерфейс взаємодії з користувачем, PicoClaw – як фреймворк для організації логіки AI-агента, а LM Studio – як середовище локального запуску LLM. LM Studio може працювати як локальний менеджер мовних моделей і надавати OpenAI-сумісний API; за замовчуванням локальна адреса сервера має вигляд http://localhost:1234/v1, а витрати не формуються за принципом оплати кожного токена, як у хмарних API. У запропонованому варіанті LM Studio розгортається на вже наявній робочій станції (наприклад, 2 CPU Intel Xeon E5-2673v4 та GPU RTX 3060 12 GB) [43]. При використанні безкоштовного ПЗ, то основними є витрати на розробку, електроенергію та супровід:

$$C_{\Sigma} = C_{dev} + C_{el} + C_{sup}, \tag{3.1}$$

де C_{dev} – витрати на розробку та налаштування;

C_{el} – витрати на електроенергію;

C_{sup} – витрати на супровід системи.

Витрати на розробку визначаються через трудомісткість виконаних робіт. Сюди входить створення Telegram-бота, налаштування PicoClaw, підключення LM Studio API, завантаження та тестування локальної LLM, перевірка повного циклу взаємодії: повідомлення користувача в Telegram – передача запиту до PicoClaw – генерація відповіді в LM Studio – повернення відповіді в чат. Наприклад, на створення прототипу було витрачено 35 годин, а умовна вартість однієї години роботи становить 150 грн:

$$C_{dev} = T_{dev} \cdot R_{dev} = 35 \cdot 150 = 5250 \text{ грн}, \tag{3.2}$$

де T_{dev} – кількість годин, витрачених на розробку, налаштування та тестування AI-агента;

R_{dev} – умовна вартість однієї години роботи розробника.

Надалі вважаємо, що середня потужність споживання для робочої станції становить ≈ 0,336 кВт.

Станом на 2026 р. для побутових споживачів в Україні діє тариф 4,32 грн/кВт·год. Для прикладу приймемо, що AI-агент працює 5 годин на день протягом 20 робочих днів на місяць:

$$C_{el} = P_{avg} \cdot T_{work} \cdot K_{el} = 0{,}336 \cdot 5 \cdot 20 \cdot 4{,}32 = 145 \text{ грн/ міс}, \quad (3.3)$$

де P_{avg} – середня потужність системи, кВт;

T_{work} – тривалість роботи системи за місяць, год;

K_{el} – тариф за 1 кВт·год.

Витрати на супровід системи оцінюємо через витрачений час на перевірку роботи бота, оновлення моделі, перезапуск LM Studio, коригування конфігурації PicoClaw або аналіз помилок. Якщо прийняти, що на супровід витрачається 2 години на місяць, а вартість однієї години становить 150 грн, то отримаємо таке значення:

$$C_{sup} = T_{sup} \cdot R_{sup} == 2 \cdot 150 = 300 \text{ грн/міс}, \quad (3.4)$$

де T_{sup} – кількість годин супроводу на місяць;

R_{sup} – умовна вартість однієї години супроводу.

Тоді щомісячні експлуатаційні витрати після створення системи становитимуть:

$$C_{month} = C_{el} + C_{sup} = 145 + 300 = 445 \text{ грн/міс}. \quad (3.5)$$

Повні витрати за перший місяць з урахуванням розробки можна оцінити так:

$$C_{1\Sigma} = C_{month} + C_{dev} = 5250 + 445 = 5695 \text{ грн}. \quad (3.6)$$

У наступні місяці, коли система вже налаштована, основні витрати зменшуються до експлуатаційного рівня: ≈445 грн/місяць за прийнятих умов. Якщо прийняти, що AI-агент обробляє 3000 повідомлень на місяць, то:

$$C_{req} = \frac{C_{month}}{N} = \frac{445}{3000} \approx 0{,}148 \text{ грн/запит}, \quad (3.7)$$

де C_{req} – умовна собівартість одного запиту;

N – кількість оброблених запитів за місяць.

Локальна реалізація AI-асистента на базі LM Studio є економічно доцільною для тестового застосування, оскільки після початкового налаштування система не потребує оплати кожного звернення до API хмарної LLM. Основні витрати формуються за рахунок електроенергії та мінімального супроводу.

Додатковий економічний ефект може бути пов’язаний зі скороченням часу на обробку типових повідомлень користувачів. Якщо припустити, що без AI-агента оператор витрачав би в середньому 2 хвилини на одне звернення, то зекономлений час для 3000 звернень на місяць можна визначити за формулою:

$$T_{save} = N \cdot T_{avg} = 3000 \cdot 2 = 6000 \text{ хвилин}, \quad (3.8)$$

де T_{save} – загальний зекономлений час;

T_{avg} – середній час ручної обробки одного звернення.

Якщо умовна вартість однієї години роботи спеціаліста становить $R = 100$ грн, то грошовий еквівалент зекономленого часу можна оцінити так:

$$C_{time} = T_{save} \cdot R = 100 \cdot 100 = 10000 \text{ грн/міс}. \quad (3.9)$$

Чистий умовний економічний ефект можна визначити за виразом:

$$E = C_{time} - C_{month} = 10000 - 445 = 9555 \text{ грн/міс}. \quad (3.10)$$

Хоча розрахунок є орієнтовним (не всі відповіді AI-агента можуть повністю замінити роботу людини), все ж таки він демонструє практичну економічну доцільність у завданнях автоматизації інформаційних запитів.

ВИСНОВКИ

AI-агент може виконувати послідовність дій для досягнення певної мети. Агентні системи поступово переходять від етапу експериментів до практичного використання. Основними тенденціями є розвиток спеціалізованих агентів, Computer Use, MCP та локальних AI-рішень.

Фреймворки AI-агентів спрощують створення агентних систем. Вони надають готові засоби для роботи з LLM, пам’яттю, інструментами та плануванням. Це дозволяє не створювати всю логіку агента з нуля. Водночас велика кількість рівнів абстракції може ускладнювати розуміння роботи системи. Фреймворки AI-агентів можуть мати різні типи архітектур. До них належать одноагентні, workflow-, graph-, multi-agent, RAG-based, human-in-the-loop та local/edge архітектури. На практиці часто поєднують кілька архітектур одночасно. Не всі задачі потребують повноцінного агентного підходу. Тому такі фреймворки доцільно застосовувати лише тоді, коли вони справді дають практичну перевагу.

Фреймворки AI-агентів мають не лише переваги, а й певні обмеження. До них належать нестабільність роботи, складність контролю, висока вартість обчислень і надмірна складність архітектури. Під час вибору фреймворку важливо враховувати його функції, складність, підтримку інструментів і можливість практичного використання. Також потрібно оцінювати обмеження, вартість і рівень контролю над діями AI-агента. Вибір інструментарію має враховувати не лише функціональність. Важливими є безпека, конфіденційність, права доступу та вартість використання. Для критичних дій доцільно залишати підтвердження з боку людини. Це зменшує ризики помилкових або небажаних дій агента.

Впровадження agentic AI доцільно починати з одного конкретного робочого процесу. Це дозволяє зменшити ризики та простіше оцінити результат. Важливо заздалегідь визначити вхідні дані, очікувані результати, інструменти та можливі помилки. Поступове тестування дає змогу безпечно

перейти до практичного використання AI-агента. Структуровані інструкції є важливою частиною роботи AI-агента. Вони задають його роль, цілі, правила поведінки та доступні інструменти. Чим точніше сформульовані інструкції, тим стабільнішою є робота агента. AI-агенти можуть мати різні функції. Для багатьох задач достатньо одного агента з набором інструментів. Multi-agent підхід доцільний тоді, коли завдання є складним і потребує розподілу ролей. Найкращим підходом є поступове ускладнення системи лише за потреби.

Локальний AI-агент є перспективним рішенням для задач, де важливі конфіденційність і контроль над даними. Такий агент може працювати з локальною LLM, файлами, базами знань та інструментами без обов'язкового використання хмарних сервісів. Для його реалізації можна використовувати «легкі» фреймворки (PicoClaw, PicoFlow, PocketFlow або n8n).

Локальна реалізація AI-агента може бути економічно доцільною. Після початкового налаштування система не потребує оплати кожного запиту до хмарної LLM. Основні витрати пов'язані з електроенергією, розробкою та супроводом. Такий підхід є корисним для автоматизації типових інформаційних запитів. Експериментальна реалізація показала можливість створення AI-агента на основі PicoClaw та LM Studio. Такий підхід демонструє роботу простого локального AI-асистента без обов'язкового використання хмарного LLM-сервісу.

Таким чином, результатами роботи є узагальнена класифікація типових архітектур фреймворків AI-агентів, модель локального AI-агента на основі PicoClaw, рекомендації щодо практичної реалізації AI-агента. Вони можуть бути використані для подальших досліджень за даною тематикою та при проектуванні локальних AI-агентів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Slyusar V., Sliusar I. Leveraging pre-trained neural networks for image classification in audio signal analysis for mobile applications of home automation. *Research Tendencies and Prospect Domains for AI Development and Implementation*. River Publishers, 2024. P. 109-128.

2. Huang K. Agentic AI: Theories and Practices. *Cham: Springer*, 2025. 407 p. DOI: https://doi.org/10.1007/978-3-031-90026-6.

3. Why Large Language Models are the future of manufacturing. *Weforum*. URL: https://www.weforum.org/stories/2024/04/why-large-language-models-are-so-important-for-the-future-of-the-manufacturing-industry/ (дата звернення: 05.05.2026)

4. LangGraph overview. *LangChain*. URL: https://docs.langchain.com/ oss/python/langgraph/overview (дата звернення: 05.05.2026)

5. Agent and Multi-Agent Applications. *Microsoft*. URL: https://microsoft.github.io/autogen/stable//user-guide/core-user-guide/core-concepts/agent-and-multi-agent-application.html (дата звернення: 05.05.2026)

6. Flows. *CrewAI*. URL: https://docs.crewai.com/en/concepts/flows (дата звернення: 05.05.2026)

7. LlamaIndex. URL: https://www.llamaindex.ai (дата звернення: 05.05.2026)

8. The Open Source AI Framework for Production Ready Agents, RAG & Context Engineering. *Haystack*. URL: https://haystack.deepset.ai (дата звернення: 05.05.2026)

9. Get Work done with SuperAGI. SuperAGI. URL: https://superagi.com (дата звернення: 05.05.2026)

10. AgentGPT. *Reworkd AI*. URL: https://agentgpt.reworkd.ai (дата звернення: 05.05.2026)

11. Maheshus M. Retrieval-Augmented Generation (RAG): Real Advances in 2025. Medium.com. URL: https://medium.com/@maheshus007/ retrieval-

augmented-generation-rag-real-advances-in-2025-8a0933ce20c2 (дата звернення: 05.05.2026)

12. Structured model outputs: JSON mode. *OpenAI*. URL: https://developers.openai.com/api/docs/guides/structured-outputs#json-mode (дата звернення: 05.05.2026)

13. AI agent orchestration patterns. *Azure Architecture Center*. URL: https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/ai-agent-design-patterns (дата звернення: 05.05.2026)

14. Model Context Protocol. Specification. *Version 2025-03-26 Model Context Protocol*. URL: https://modelcontextprotocol.io/specification/2025-03-26 (дата звернення: 05.05.2026)

15. Biswas A., Talukdar W. Building Agentic AI Systems. Birmingham: Packt Publishing, 2025. 292 p.

16. Sapkota R., Roumeliotis K., Karkee M. AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenges. URL: https://arxiv.org/abs/2505.10468 (дата звернення: 05.05.2026)

17. Computer use. *OpenAI API Documentation*. URL: https://developers.openai.com/api/docs/guides/tools-computer-use (дата звернення: 05.05.2026)

18. What are projects? *Anthropic*. URL: https://support.anthropic.com/en/articles/9517075-what-are-projects (дата звернення: 05.05.2026)

19. Gemma models overview. Google AI for Developers. *Google*. URL: https://ai.google.dev/gemma/docs (дата звернення: 05.05.2026)

20. Vertex AI Agent Builder documentation. *Google Cloud*. URL: https://docs.cloud.google.com/agent-builder (дата звернення: 05.05.2026)

21. Microsoft Copilot Studio documentation. *Microsoft.* URL: https://learn.microsoft.com/en-us/microsoft-copilot-studio/ (дата звернення: 05.05.2026)

22. Introducing Zapier Central: Teach AI bots to work on their own across your apps. *Zapier*. URL: https://zapier.com/blog/introducing-zapier-central-ai-bots/ (дата звернення: 05.05.2026)

23. Perplexity Pages. Perplexity Help Center. *Perplexity*. URL: https://www.perplexity.ai/help-center/en/articles/10352968-perplexity-pages (дата звернення: 05.05.2026)

24. Introducing our new mobile browser, Arc Search. *The Browser Company*. URL: https://arc.net/blog/arc-search (дата звернення: 05.05.2026)

25. About Autopilot UiPath Documentation. *UiPath*. URL: https://docs.uipath.com/autopilot/other/latest/user-guide/about-autopilot (дата звернення: 05.05.2026)

26. GitHub Copilot Workspace: Welcome to the Copilot-native developer environment. *GitHub*. URL: https://github.blog/news-insights/product-news/github-copilot-workspace/ (дата звернення: 05.05.2026)

27. What is the Model Context Protocol (MCP)? *Model Context Protocol*. URL: https://modelcontextprotocol.io/docs/getting-started/intro (дата звернення: 05.05.2026)

28. Agents. *CrewAI*. URL: https://docs.crewai.com/en/concepts/agents (дата звернення: 05.05.2026)

29. OpenAI Agents SDK. Handoffs. *OpenAI*. URL: https://openai.github.io/openai-agents-python/handoffs/ (дата звернення: 05.05.2026)

30. n8n Cloud. *n8n*. URL: https://docs.n8n.io/manage-cloud/overview/ (дата звернення: 05.05.2026)

31. Make.com Documentation. *Make.com*. URL: https://www.make.com/en/help (дата звернення: 05.05.2026)

32. Raspberry Pi. URL: https://www.raspberrypi.com (дата звернення: 05.05.2026)

33. LM Studio. URL: https://lmstudio.ai (дата звернення: 05.05.2026)

34. Ollama. URL: https://ollama.com (дата звернення: 05.05.2026)

35. Koboldcpp installation package. URL: https://kcpptools.concedo.workers.dev/ (дата звернення: 05.05.2026)

36. Picoclaw. URL: https://github.com/sipeed/picoclaw (дата звернення: 05.05.2026)

37. Picoflow. URL: https://github.com/the-picoflow/picoflow (дата звернення: 05.05.2026)

38. PocketFlow. URL: https://github.com/the-pocket/PocketFlow (дата звернення: 05.05.2026)

39. Вовнянко І.В. Використання n8n для створення AI-агенту. *Збірник XXI щорічної студентської наукової конференції «Сучасні інформаційні технології та інноваційні методики в економіці, менеджменті та бізнесі» Полтавського державного аграрного університету* (квітень 2026 р., м. Полтава). Полтава: ПДАУ, 2026 р. С. 68-70.

40. Pappe F. n8n: Build a Local AI Agent for free. *Medium.* URL: https://medium.com/data-science-collective/n8n-free-local-ai-agent-with-ollama-82d70c1915f2 (дата звернення: 05.05.2026)

41. Docker. URL: https://www.docker.com (дата звернення: 05.05.2026)

42. The AI Reliability Platform. Guardrails AI. URL: https://guardrailsai.com (дата звернення: 05.05.2026)

43. Двопроцесорна робоча станція PowerUp #201 Xeon E5 2673 v4 x2/64 GB/SSD 512GB/GeForce RTX 3060 12GB URL: https://powerup.ua/dvukhprotsessornaya-rabochaya-stantsiya-powerup-345-xeon-e5-2673-v4-x2-64-gb-ssd-480-gb-geforce-rtx-3060-12gb/ (дата звернення: 05.05.2026)