Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Метод динамічного планування для LLM-інтегрованих агентів на основі адаптивного синтезу політик
(тема)

Виконав:
здобувач другого року навчання,
групи СШІм-24-1
Володимир МАХОТІЛО
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва освітньої програми)
Керівник проф. Олександр ШЕВЧЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Лариса ЧАЛА
(підпис) (власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Махотілу Володимиру Костянтиновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Метод динамічного планування для LLM-інтегрованих агентів на основі адаптивного синтезу політик

затверджена наказом університету від 6 квітня 20 26 р. № 269Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 21 травня 20 26 р.

3. Вихідні дані до роботи Веб-середовище WebArena, веб-середовище MiniWobPlusPlus, документація до WebArena, документація до MiniWobPlusPlus, документація з бібліотек: JSON, OpenAI, NumPy, requests, дослідницькі матеріали методів вирішення задач веб-навігації, дослідницькі матеріали з методів вирішення задач довгогоризонтного планування, дослідницькі матеріали з методів побудови мультиагентних систем.

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задачі
2) Теоретичні дослідження
3) Практичні дослідження

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	06.04.2026	виконано
2	Аналіз предметної галузі	07.04.2026 - 16.04.2026	виконано
3	Постановка задачі	17.04.2026 - 19.04.2026	виконано
4	Теоретичні дослідження	20.04.2026 - 01.05.2026	виконано
5	Практичні дослідження	02.05.2026 - 09.05.2026	виконано
6	Написання пояснювальної записки	10.05.2026 - 17.05.2026	виконано
7	Створення презентації	18.05.2026 - 20.05.2026	виконано
8	Захист перед ЕК	21.05.2026	виконано

Дата видачі завдання 6 квітня 20 26 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Олександр ШЕВЧЕНКО
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 91 с., 4 рис., 1 табл., 4 дод., 19 джерел.

АГЕНТ, ВЕБ-АГЕНТ, ВЕЛИКА МОВНА МОДЕЛЬ, ДИНАМІЧНЕ ПЛАНУВАННЯ, ЗАДАЧІ ВЕБ-НАВІГАЦІЇ, ЗАДАЧІ ДОВГОГОРИЗОНТНОГО ПЛАНУВАННЯ, ІНТЕРАКТИВНЕ СЕРЕДОВИЩЕ, ТОКЕН.

Об'єкт дослідження – застосування методів динамічного планування для LLM-інтегрованих агентів із використанням адаптивного синтезу політик у задачах довгогоризонтного планування, що виникають у процесі взаємодії з інтерактивними середовищами, зокрема в рамках задач веб-навігації.

Предмет дослідження – методи динамічного планування для LLM-інтегрованих агентів на основі адаптивного синтезу політик, а також підходи до їх оцінювання та прототипування.

Мета роботи – визначення та імплементація методу динамічного планування для LLM-інтегрованих агентів на основі адаптивного синтезу політик.

Методи дослідження – теоретичний аналіз існуючих методів динамічного планування, розробка методу на основі синергії наявних підходів, проведення експериментів та аналіз отриманих результатів.

Результатом дослідження є проведення аналізу існуючих методів, розробка методу, що є синергією наявних підходів з адаптацією до задач веб-навігації, його програмна реалізація та подальше оцінювання ефективності.

ABSTRACT

Master’s thesis contains: 91 pp., 4 fig., 1 tabl., 4 ann., 19 references.

AGENT, DYNAMIC PLANNING, INTERACTIVE ENVIRONMENT, LARGE LANGUAGE MODEL, LONG-TERM PLANNING PROBLEMS, TOKEN, WEB AGENT, WEB NAVIGATION PROBLEMS.

Object of research – the application of dynamic planning methods for LLM-integrated agents using adaptive policy synthesis in long-horizon planning problems that arise during interaction with interactive environments, in particular within web navigation tasks.

Subject of research – dynamic planning methods for LLM-integrated agents based on adaptive policy synthesis, as well as approaches to their evaluation and prototyping.

Purpose of the work – to define and implement a dynamic planning method for LLM-integrated agents based on adaptive policy synthesis.

Research methods – theoretical analysis of existing dynamic planning methods, development of a method based on the synergy of existing approaches, conducting experiments, and analysis of the obtained results.

The result of the research is the analysis of existing methods, the development of a method that combines existing approaches and adapts them to web navigation tasks, its software implementation, and subsequent evaluation of its effectiveness.

ЗМІСТ

Вступ 8
1 Аналіз предметної галузі та постановка задачі 9
1.1 Опис предметної галузі 9
1.1.1 Розвиток штучного інтелекту і чат-орієнтованих систем 9
1.1.2 Мультиагентні системи на базі великих мовних моделей 11
1.2 Класи та типи задач LLM-інтегрованих агентів 14
1.2.1 Короткогоризонтні та довгогоризонтні задачі 14
1.2.2 Задачі веб-навігації 16
1.3 Актуальність дослідження 17
1.4 Постановка задачі 19
2 Теоретичні дослідження 20
2.1 Методи динамічного синтезу політик 20
2.1.1 Ієрархічне планування 20
2.1.2 Адаптивний синтез навичок 22
2.1.3 Few-shot prompting 24
2.1.4 Адаптивний синтез few-shot прикладів 26
2.1.5 Деревоподібний пошук траєкторій 27
2.2 Короткострокова та довгострокова пам’яті в LLM-інтегрованих агентах 28
2.2.1 Збереження історії взаємодії в контексті промпта 30
2.2.2 Підсумовування траєкторії виконання 31
2.2.3 Використання зовнішньої бази знань або векторного сховища 31
2.3 Інтерактивні середовища та набори даних для тестування систем ... 32
2.3.1 Середовище MiniWobPlusPlus 32
2.3.2 Середовище WebArena 34
2.3.3 Набір данних Mind2Web 35
2.4 Метрики оцінювання якості роботи агента 36
3 Практичні дослідження 39

3.1 Реалізовані агенти 39
3.1.1 Планувальник глобального плану 39
3.1.2 Агент спостереження та рефлексії 40
3.1.3 Агент виконання дій 41
3.1.4 Менеджер знань 43
3.2 Процес виконання задачі 44
3.3 Технічні особливості обраних тестових середовищ 46
3.3.1 Особливості середовища WebArena 46
3.3.2 Особливості середовища MiniWob 48
3.4. Проведені експерименти 49
3.4.1 Проведення експериментів на основі MiniWobPlusPlus 49
3.4.2 Спостереження під час виконання WebArena 51
3.5 Можливі напрями вдосконалення 52
3.5.1 Векторна база даних 52
3.5.2 Динамічна регенерація глобального плану 53
3.6 Порівняння з іншими моделями 54
Висновки 56
Перелік джерел посилання 58
Додаток А Вихідні дані з тестових середовищ 60
Додаток Б Текстові промпти агентів 77
Додаток В Вихідний програмний код оркестрації мультагентної системи 85
Додаток Г Відомість кваліфікаційної роботи 91

ВСТУП

Галузь штучного інтелекту є однією з найважливіших і найдинамічніших у сучасному світі. З розвитком технологій та зростанням обсягів інформації особливого значення набули системи, здатні не лише аналізувати дані, але й самостійно взаємодіяти з середовищем та приймати послідовні рішення. Одним із ключових напрямів такого розвитку стали великі мовні моделі, які демонструють здатність до узагальнення, міркування та виконання складних інструкцій.

Зі зростанням популярності великих мовних моделей зростає як кількість, так і складність задач, що можуть бути розв’язані з їх використанням. Початкові задачі, які передбачали взаємодію в межах одного запиту між агентом і користувачем, еволюціонували у більш складні сценарії, що потребують багатоетапного виконання та послідовного прийняття рішень. У таких умовах дедалі більшої актуальності набувають підходи, що передбачають динамічне формування стратегії поведінки агента.

Збільшення складності задач зумовило також ускладнення підходів до їх розв’язання. Зокрема, виникла необхідність у використанні мультиагентних систем, у яких декілька агентів взаємодіють між собою з метою декомпозиції задачі на підзадачі, паралельного виконання окремих етапів або підвищення якості кінцевого результату.

Мета роботи**:** Розробка та оцінка методу динамічного планування для LLM-інтегрованих агентів на основі адаптивного синтезу політик.

Основне завдання**:** Теоретичне дослідження наявних методів динамічного планування та практичне дослідження розробленого методу динамічного планування для LLM-інтегрованих агентів на основі адаптивного синтезу політик.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної галузі

1.1.1 Розвиток штучного інтелекту і чат-орієнтованих систем

З кожним роком сфера штучного інтелекту займає дедалі важливіше місце в сучасній технологічній індустрії та поступово впливає на все більшу кількість прикладних галузей. Хоча штучний інтелект часто сприймається як відносно новий напрям, історія його розвитку та практичного застосування налічує вже кілька десятиліть. За цей час галузь пройшла кілька важливих етапів, кожен із яких був пов’язаний із появою нових підходів, методів, архітектур і технологічних рішень.

Протягом розвитку штучного інтелекту значна кількість наукових праць, технологічних продуктів і відкриттів суттєво змінювала уявлення про можливості інтелектуальних систем. На різних етапах розвитку галузі змінювався й набір основних технологій, які вважалися найбільш перспективними. Ці зміни впливали не лише на наукові дослідження, а й на індустрію, бізнес-процеси, роботодавців і ринок праці. З появою нових підходів змінювалися вимоги до спеціалістів, формувалися нові професійні ролі та виникала потреба в нових компетентностях.

Одним із важливих етапів в історії розвитку штучного інтелекту був період експертних систем. Такі системи будувалися на основі явно заданих правил і знань, сформованих за участю фахівців певної предметної області. Їхня основна ідея полягала в тому, щоб описати процес прийняття рішень через набір логічних правил. Такий підхід був достатньо ефективним для вузьких і добре формалізованих задач, однак мав суттєві обмеження.

Головною проблемою експертних систем була складність масштабування та підтримки. Для кожної нової предметної області потрібно було вручну формувати базу знань, описувати правила, перевіряти їхню

узгодженість і регулярно оновлювати систему відповідно до змін у предметній області. У результаті такі системи погано адаптувалися до складних, динамічних або слабо формалізованих задач. Це поступово зумовило перехід до статистичних методів машинного навчання, які дозволяли не задавати всі правила вручну, а автоматично виявляти закономірності на основі даних.

Важливим чинником розвитку машинного навчання стало зростання обсягів доступних даних. Розвиток технологій зберігання, обробки та передачі інформації сприяв появі напряму Big Data, що створило умови для застосування статистичних методів у значно більшому масштабі. На відміну від експертних систем, моделі машинного навчання могли навчатися на великих наборах даних і виявляти закономірності, які складно або неможливо було явно описати у вигляді правил.

Подальший розвиток машинного навчання привів до зростання ролі глибинного навчання. Цей етап став можливим завдяки одночасному розвитку кількох факторів: збільшенню обсягів даних, зростанню доступних обчислювальних ресурсів, розвитку графічних процесорів, а також появі нових архітектур нейронних мереж. Глибинне навчання дозволило значно покращити результати в задачах комп’ютерного зору, обробки природної мови, розпізнавання мовлення та багатьох інших напрямах.

Особливо важливим етапом став розвиток методів обробки природної мови. Поява механізму уваги та архітектури Transformer суттєво змінила підходи до побудови мовних моделей. На основі цих підходів почали активно розвиватися великі мовні моделі, здатні працювати з текстом на значно вищому рівні, ніж попередні системи. Вони отримали здатність генерувати зв’язні відповіді, аналізувати великі обсяги тексту, виконувати інструкції користувача та розв’язувати широкий спектр задач без окремого навчання для кожної з них.

Кульмінацією цього розвитку стало широке впровадження чат-орієнтованих систем на основі великих мовних моделей. Такі системи зробили взаємодію зі штучним інтелектом доступною для масового користувача, оскільки дозволили працювати з моделями через природну мову без потреби у спеціальних технічних знаннях. Зокрема, поява рішень компанії OpenAI стала одним із ключових чинників популяризації великих мовних моделей і значно посилила інтерес до практичного застосування штучного інтелекту в освіті, бізнесі, програмуванні, аналітиці та інших сферах.

1.1.2 Мультиагентні системи на базі великих мовних моделей

Незадовго після стрімкого зростання популярності чат-орієнтованих великих мовних моделей почали активно з’являтися публічні програмні інтерфейси для взаємодії з ними. Це стало важливим етапом у розвитку прикладного використання LLM, оскільки мовні моделі перестали бути лише окремими чат-системами для генерації тексту. Вони поступово почали інтегруватися у різні програмні продукти, інформаційні системи, сервіси підтримки користувачів, аналітичні інструменти та системи автоматизації. Одним із перших і найбільш впливових прикладів став API, запропонований OpenAI, який суттєво спростив інтеграцію мовних моделей у прикладні системи. Завдяки такому підходу розробники отримали можливість використовувати мовну модель як окремий інтелектуальний компонент, до якого можна звертатися з програмного коду.

На початкових етапах використання великих мовних моделей більшість прикладних сценаріїв будувалася навколо простого принципу: користувач або система формує запит, модель обробляє його та повертає відповідь. Такий підхід добре працює для задач, пов’язаних із генерацією тексту, перекладом, поясненням понять, класифікацією або коротким аналізом інформації. Однак зі збільшенням кількості та складності

прикладних задач стало очевидним, що базові підходи, які передбачають одноразовий запит до моделі, є недостатніми для розв'язання задач, що потребують багатоетапного міркування. У складніших випадках моделі необхідно не лише дати відповідь, а й послідовно аналізувати ситуацію, уточнювати проміжні результати, звертатися до зовнішніх інструментів і коригувати подальші дії залежно від отриманих результатів.

Такі обмеження зумовили появу нових підходів до організації роботи LLM, зокрема агентних архітектур. На відміну від звичайного використання мовної моделі як генератора відповіді, агентна архітектура передбачає, що модель стає частиною ширшої системи прийняття рішень. У такій системі LLM може виконувати роль планувальника, інтерпретатора інструкцій, генератора дій або модуля оцінювання результатів. Агент отримує початкову задачу, аналізує поточний стан середовища, обирає наступну дію, виконує її через доступні інструменти або інтерфейси, після чого аналізує нове спостереження. Таким чином, робота системи набуває ітеративного характеру.

Одним із ключових кроків у цьому напрямі стала поява підходу ReAct [1], який інтегрує процеси міркування та виконання дій у єдиний ітеративний цикл. У межах цього підходу агент не лише генерує текстові відповіді, але й формує проміжні кроки мислення, які використовуються для прийняття подальших рішень. Ідея ReAct полягає в тому, що міркування та дія не повинні існувати окремо. Агент спочатку аналізує поточну ситуацію, потім обирає дію, виконує її та отримує нове спостереження. Після цього цикл повторюється. Така структура дозволяє агенту поступово просуватися до розв'язання задачі, а не намагатися сформувати повну відповідь за один запит.

Перевагою такого підходу є те, що агент може реагувати на зміну стану середовища [2]. Наприклад, якщо виконана дія не дала очікуваного результату, агент може змінити стратегію, повернутися до попереднього кроку або сформувати новий план. Це особливо важливо для задач, у яких

результат кожної наступної дії залежить від попередніх дій. До таких задач можна віднести веб-навігацію, роботу з програмними інтерфейсами, пошук інформації, планування дій у середовищі або виконання складних інструкцій користувача.

Подальший розвиток агентних підходів привів до формування мультиагентних систем на базі LLM [3]. У таких системах замість одного універсального агента використовується набір спеціалізованих агентів, кожен з яких відповідає за окрему підзадачу або функціональну роль. Наприклад, можуть існувати окремі агенти для планування, виконання дій, перевірки результатів та рефлексії (рисунок 1.1).

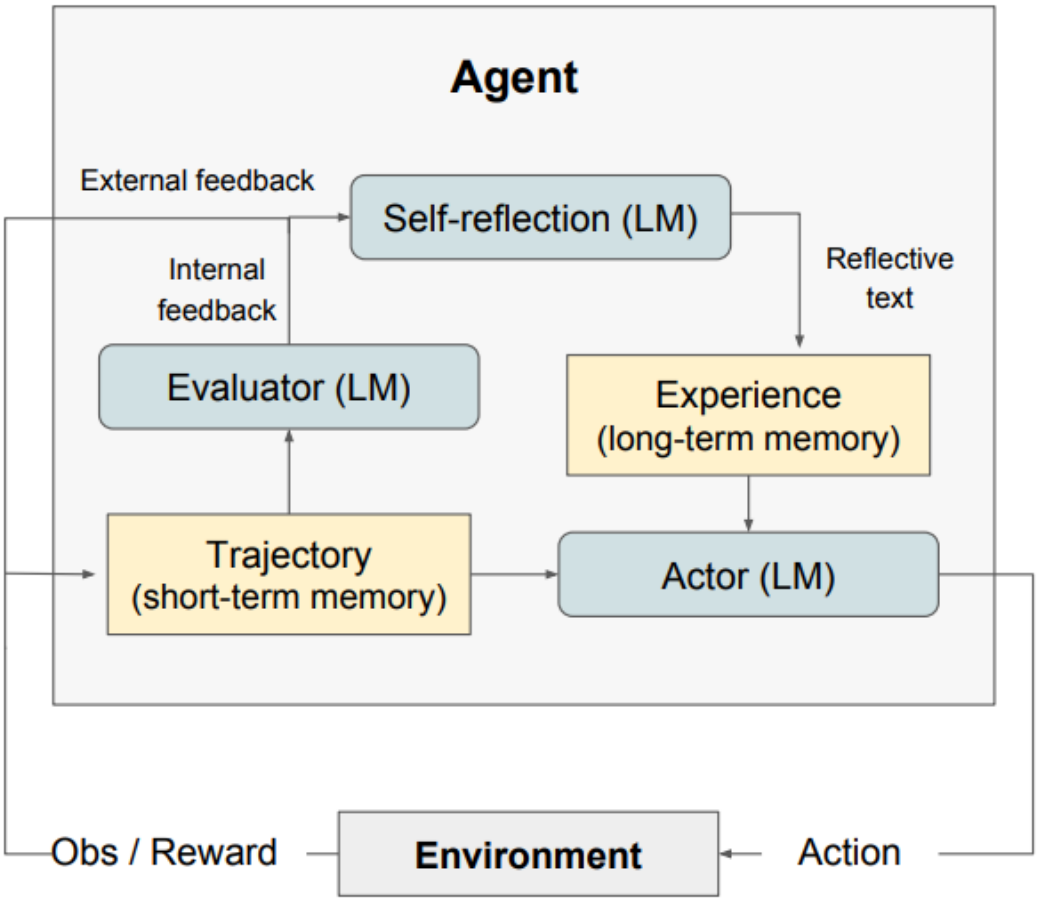


Рисунок 1.1 – Мультиагентна архітектура з механізмом саморефлексії

1.2 Класи та типи задач LLM-інтегрованих агентів

LLM-інтегровані агентні системи застосовуються для вирішення широкого спектра задач, у яких мовна модель використовується не лише як засіб генерації тексту, а як центральний компонент прийняття рішень. У таких системах LLM може аналізувати вхідний запит, формувати проміжні міркування, обирати подальші дії, взаємодіяти із зовнішніми інструментами або середовищем, а також коригувати власну поведінку на основі отриманих результатів.

Залежно від характеру задачі, LLM-інтегровані агенти можуть використовуватися у різних прикладних напрямах. Найбільш поширеними класами задач є:

– задачі інформаційного пошуку та відповіді на запити відносно бази знань;

– задачі генерації та модифікації генерацію тексту;

– задачі генерації та модифікації програмного коду;

– задачі генерації та модифікації зображань;

– задачі генерації та модифікації відео контенту;

– задачі генерації та модифікації аудіо контенту;

– задачі інтерактивної взаємодії з середовищем, що передбачають вирішення задачі, через взаємодію з середовищем.

1.2.1 Короткогоризонтні та довгогоризонтні задачі

За рівнем складності процесу розв’язання задачі LLM-інтегровані агентні системи можна умовно поділити на два класи: короткогоризонтні та довгогоризонтні задачі [4]:

– короткогоризонтні задачі – задачі, що можуть бути розв’язані агентом у межах одного запиту або кількох незалежних запитів, не

потребують виконання тривалих послідовностей дій, відстеження траєкторії виконання та складного планування;

– довгогоризонтні задачі – задачі, що характеризуються необхідністю виконання тривалих послідовностей взаємопов’язаних дій, де кожне наступне рішення залежить від попередніх станів середовища.

Довгогоризонтні та короткогоризонтні задачі є достатньо широкими класами, при цьому належність конкретної задачі до одного з них не завжди є однозначною. Деякі типи задач можуть відноситися як до короткогоризонтних, так і до довгогоризонтних залежно від складності постановки та способу їх розв’язання.

До короткогоризонтних задач належать:

– виконання простих обчислювальних операцій;

– інформаційний пошук;

– генерація тексту.

До довгогоризонтних задач належать:

– багатоетапні обчислювальні процеси;

– багатоетапний інформаційний пошук;

– керування роботизованими системами;

– навігація у віртуальних або ігрових середовищах;

– веб-навігація та взаємодія з веб-інтерфейсами.

Важливим класом довгогоризонтних задач є задачі, що передбачають взаємодію з інтерактивним середовищем [5]. На відміну від задач, у яких агент лише формує відповідь на основі вхідного запиту, у таких задачах результат досягається через послідовне виконання дій. Агент не має повного контролю над середовищем і після кожної дії отримує нове спостереження, яке може підтверджувати правильність обраного напряму або, навпаки, вказувати на помилку, зміну стану чи необхідність корекції подальшого плану.

Особливість таких задач полягає в тому, що кожна дія агента впливає на подальший процес виконання. Наприклад, натискання кнопки, введення

даних у форму, перехід на іншу сторінку або вибір певного елемента інтерфейсу змінюють стан середовища. Після цього агент повинен повторно проаналізувати ситуацію та визначити, чи наблизила його попередня дія до досягнення кінцевої мети. Таким чином, процес розв’язання задачі набуває ітеративного характеру.

У довгогоризонтних задачах важливим стає не лише правильний вибір окремої дії, а й узгодженість усієї послідовності дій. Окрема дія може бути формально правильною, але неефективною у межах загальної стратегії. Наприклад, агент може правильно перейти на певну сторінку, але зробити це у невідповідний момент або без урахування попередніх результатів. Через це для таких задач необхідно зберігати історію виконання, аналізувати попередні кроки та контролювати відповідність поточного стану початковій цілі.

Також для довгогоризонтних задач характерною є наявність помилок, тупикових ситуацій і циклів. Агент може повторювати однакові дії, неправильно інтерпретувати стан середовища або втрачати зв’язок між локальною дією та глобальною метою. У таких випадках простого покрокового виконання інструкцій недостатньо. Система повинна мати можливість оцінювати власну траєкторію, виявляти неефективні дії та змінювати план виконання.

1.2.2 Задачі веб-навігації

Одним із найбільш показових прикладів довгогоризонтних задач є задача веб-навігації [6]. Під веб-навігацією в контексті LLM-інтегрованих агентів розуміється процес взаємодії агента з веб-середовищем з метою виконання певної користувацької задачі. Така взаємодія може включати відкриття сторінок, натискання кнопок, введення тексту, вибір елементів інтерфейсу, роботу з формами, перехід між сторінками та аналіз отриманої інформації.

Задачі веб-навігації є складними, оскільки веб-середовище має динамічний характер. Сторінка може змінюватися після кожної дії агента: відкриваються нові елементи, з’являються повідомлення про помилки, змінюється URL, оновлюється вміст або стан форми. Тому агент повинен не лише виконувати інструкцію користувача, а й постійно аналізувати, чи наближає поточна дія систему до фінальної цілі, чи дійсно була успішна проведена взаємодія з елементом.

Формально задачу веб-навігації можна описати як послідовність станів, дій та спостережень. На кожному кроці агент отримує поточне спостереження про стан веб-сторінки, обирає дію та виконує її. Після цього середовище переходить у новий стан, а агент отримує нове спостереження. Цей процес повторюється до моменту досягнення цілі або завершення виконання через помилку чи обмеження кількості кроків.

1.3 Актуальність дослідження

Тема адаптивного синтезу політик є актуальною у зв’язку з тим, що значна кількість сучасних задач не може бути ефективно розв’язана за допомогою наперед заданих статичних правил або фіксованих стратегій поведінки [7]. У класичних підходах до побудови інтелектуальних систем зазвичай передбачається, що політика прийняття рішень визначається ще до початку виконання задачі. Тобто система має певний набір правил, алгоритмів або інструкцій, відповідно до яких вона повинна діяти в усіх можливих ситуаціях. Такий підхід може бути достатньо ефективним у простих або добре формалізованих задачах, де середовище є стабільним, кількість можливих станів обмежена, а послідовність дій може бути передбачена заздалегідь.

Однак у більш складних інтерактивних середовищах використання статичних політик має суттєві обмеження. Це пов’язано з тим, що реальний процес виконання задачі часто є непередбачуваним. Агент може отримувати

нові спостереження, стикатися з неочікуваними станами середовища, помилками інтерфейсу, неповною інформацією або необхідністю змінити початковий план дій. У таких умовах заздалегідь визначена політика може швидко втрачати актуальність, оскільки вона не враховує зміну контексту під час виконання задачі. Внаслідок цього система може продовжувати виконувати дії, які вже не відповідають поточній ситуації, або не мати механізму для корекції власної поведінки.

Особливо помітною ця проблема є у задачах з великим простором можливих станів. У таких задачах неможливо наперед описати всі потенційні ситуації, в яких може опинитися агент. Наприклад, у веб-середовищах одна й та сама ціль може вимагати різних послідовностей дій залежно від структури сторінки, доступних елементів інтерфейсу, результатів попередніх кроків або помилок, які виникли під час взаємодії. Тому ефективна система повинна не просто виконувати фіксований алгоритм, а мати здатність аналізувати поточний стан, порівнювати його з поставленою метою, змінювати план дій і формувати нову локальну стратегію залежно від отриманих спостережень.

У таких умовах LLM-інтегровані агенти стикаються з низкою характерних проблем. Однією з них є виникнення галюцинацій, коли модель формує припущення про стан середовища або доступні дії без достатнього підтвердження з боку фактичних спостережень. Іншою проблемою є потрапляння у тупикові стани, коли агент виконав певну послідовність дій, але більше не може ефективно просуватися до мети. Також поширеною проблемою є зациклення, коли система повторює одні й ті самі або дуже схожі дії, не отримуючи нового корисного результату. Це особливо критично для довгогоризонтних задач, оскільки кожна зайва або помилкова дія збільшує довжину траєкторії, витрати токенів і ймовірність подальшої деградації плану.

Саме тому використання адаптивного підходу до синтезу політик є важливим напрямом для підвищення ефективності LLM-агентів. У межах

такого підходу політика поведінки не розглядається як повністю фіксована інструкція, а формується або уточнюється в процесі виконання задачі. Агент може враховувати початкову мету, поточний стан середовища, історію попередніх дій, отримані спостереження, помилки та проміжні результати. Це дозволяє системі більш гнучко реагувати на зміну ситуації, уникати повторення неефективних дій і коригувати власну стратегію залежно від контексту.

1.4 Постановка задачі

Враховуючи аналіз, наведений у попередніх розділах, можна поставити наступні задачі до ходу роботи:

− провести аналіз існуючих методів динамічного планування для LLM-інтегрованих агентів на основі адаптивного синтезу політик;

− провести аналіз існуючих середовищ та наборів даних даних для оцінки роботи LLM-інтегрованих агентів;

− провести аналіз існуючих метрик оцінки роботи LLM-інтегрованих агентів;

− програмно реалізувати визначений метод динамічного планування для LLM-інтегрованих агентів на основі адаптивного синтезу політик;

− провести прототипування розробленого методу у мінамілистичного середовищі для базової перевірки працездатності та базових параметрів;

− провести фінальне оцінювання ефективності методу у реалістичному середовищі;

− виконати аналіз отриманих результатів та визначити ефективність запропонованого підходу у задачах довгогоризонтного планування.

2 ТЕОРЕТИЧНІ ДОСЛІДЖЕННЯ

2.1 Методи динамічного синтезу політик

2.1.1 Ієрархічне планування

Однією з ключових проблем задач довгогоризонтного планування є їх багатокроковий характер. У таких задачах результат не може бути отриманий за одну дію або за один запит до моделі, оскільки агент повинен виконати послідовність взаємопов’язаних кроків. При цьому кожна дія впливає на подальший стан середовища, а помилка на одному з етапів може призвести до неправильного виконання всієї задачі. Додатковою складністю є висока абстрактність початкової інструкції. Часто користувач формулює лише загальну ціль, не описуючи конкретну послідовність дій, яку потрібно виконати для її досягнення.

Ще однією проблемою є те, що початкова інструкція може частково втрачати актуальність у процесі виконання задачі. Це пов’язано з тим, що стан середовища змінюється після кожної дії агента. Наприклад, у задачах веб-навігації після переходу на нову сторінку, заповнення форми або натискання кнопки агент отримує новий контекст, який може вимагати уточнення попереднього плану. Тому фіксована послідовність дій не завжди є ефективною. Крім того, більшість довгогоризонтних задач є неатомарними, тобто їх неможливо розглядати як одну просту операцію. Вони потребують поділу на окремі логічні етапи, кожен з яких має власну проміжну ціль.

Ієрархічне планування спрямоване на вирішення цих проблем шляхом поділу складної задачі на підзадачі різного рівня абстракції [8]. Основна ідея такого підходу полягає в тому, що агент не намагається одразу сформувати повну послідовність низькорівневих дій для всієї задачі. Замість цього спочатку визначається загальна структура розв’язання, після чого окремі

етапи деталізуються вже під час виконання. Це дозволяє зберігати загальний напрям руху до цілі, але водночас залишати можливість для адаптації до поточного стану середовища.

Найбільш поширеним варіантом ієрархічного планування є дворівнева схема, яка включає глобальний та локальний рівні планування. Глобальний план відповідає за загальну стратегію виконання задачі. Він формується на основі початкової інструкції користувача та визначає основні етапи, які потрібно пройти для досягнення кінцевого результату. Такий план зазвичай описує задачу на високому рівні абстракції та не містить конкретних атомарних дій, наприклад натискання певної кнопки або введення тексту в конкретне поле.

Глобальний план можна розглядати як переформульовану початкову задачу, декомпозовану на набір підзадач високого рівня. Його основна функція полягає у збереженні початкової цілі протягом усього процесу виконання. Це особливо важливо для довгих траєкторій, оскільки агент може поступово втрачати контекст або надмірно фокусуватися на локальних діях. Наявність глобального плану дозволяє системі періодично перевіряти, чи відповідають поточні дії загальному напрямку розв’язання задачі.

Локальний план, на відміну від глобального, формується для виконання конкретної підзадачі. Він має більш прикладний характер і складається з атомарних або близьких до атомарних дій, які агент повинен виконати в поточному стані середовища. Наприклад, якщо глобальний план передбачає етап пошуку необхідної інформації на веб-сайті, то локальний план може включати відкриття сторінки пошуку, введення запиту, аналіз результатів і перехід за релевантним посиланням.

Важливою особливістю локального плану є його залежність від поточного стану середовища. На відміну від глобального плану, який формується на початку і залишається відносно стабільним, локальний план може змінюватися після кожного нового спостереження. Якщо агент

виконав дію і отримав неочікуваний результат, локальний план може бути переглянутий або повністю сформований заново. Саме це забезпечує адаптивність системи та дозволяє їй реагувати на помилки, зміни інтерфейсу, відсутність очікуваних елементів або появу нової інформації.

Ієрархічне планування створює умови для адаптивного синтезу політик, де рішення формуються з урахуванням як довгострокових цілей, так і поточного стану системи.

2.1.2 Адаптивний синтез навичок

Важливою особливістю великих мовних моделей є варіативність їхніх відповідей. Навіть за однакової початкової інструкції та подібного стану середовища модель може згенерувати різні варіанти міркування, вибрати іншу дію або сформувати інший план виконання задачі. З одного боку, така властивість є корисною, оскільки вона дозволяє моделі бути гнучкою та знаходити нестандартні варіанти розв’язання. З іншого боку, у задачах довгогоризонтного планування надмірна варіативність може призводити до нестабільної поведінки агента, повторення помилок або вибору неефективних дій.

У цьому контексті важливою стає здатність агента зберігати та повторно використовувати попередній досвід. Якщо система вже виконувала подібну задачу або стикалася з подібним станом середовища, доцільно не починати процес планування повністю з нуля, а використати накопичені знання. Це дозволяє зменшити кількість випадкових або неузгоджених рішень, скоротити довжину траєкторії виконання та підвищити загальну якість прийняття рішень. Саме на вирішення цієї проблеми спрямований адаптивний синтез навичок [9].

Адаптивний синтез навичок можна розглядати як підхід, у межах якого агент не лише виконує окремі задачі, але й накопичує узагальнений досвід їх виконання. У процесі взаємодії із середовищем агент формує

траєкторії, які містять інформацію про початковий запит, послідовність виконаних дій, отримані спостереження, проміжні результати, помилки та фінальний результат. Після завершення виконання така траєкторія може бути проаналізована з метою визначення, чи містить вона корисний досвід, який можна використати в майбутньому.

У випадку успішного виконання задачі система може виділити з траєкторії певну навичку. Під навичкою в цьому контексті можна розуміти узагальнений спосіб дії, який може бути повторно застосований у подібних ситуаціях. Це не обов'язково повна послідовність конкретних дій, оскільки вона може бути занадто залежною від окремого середовища. Більш доцільно розглядати навичку як опис корисної стратегії, правила поведінки або послідовності кроків, які допомогли досягти результату в певному класі задач.

Наприклад, якщо агент у веб-середовищі вже визначив, що для взаємодії з певним типом елементів інтерфейсу краще використовувати не прямий клік, а послідовність клавіш клавіатури, така інформація може бути збережена як окрема навичка. У майбутньому, коли агент знову зіткнеться з подібною ситуацією, він зможе використати це знання замість повторного пошуку рішення методом спроб і помилок. Таким чином, база навичок виконує роль довгострокової пам'яті системи.

Задля реалізації цього підходу до складу системи вводиться окремий агент або окремий логічний модуль, який відповідає за роботу з навичками. Його функції можна умовно поділити на два основні напрями. Перший напрям – це прийняття рішення щодо збереження нового знання у базі навичок. Після завершення виконання задачі та аналізу траєкторії система повинна визначити, чи є отриманий досвід достатньо корисним, узагальненим і придатним для повторного використання. Якщо так, на його основі формується нова навичка, яка додається до бази знань.

Другий напрям – це прийняття рішення щодо використання вже наявної навички під час виконання нової задачі. На кожному етапі агент

може перевіряти, чи існує у базі навичок знання, релевантне поточному стану або підзадачі. Якщо така навичка знайдена, агент може використати її як підказку для формування локального плану або вибору наступної дії. У цьому випадку навичка не замінює повністю процес прийняття рішень, а виступає додатковим джерелом контексту, яке допомагає агенту діяти більш стабільно та обґрунтовано.

2.1.3 Few-shot prompting

Few-shot prompting є одним із поширених підходів до керування поведінкою великих мовних моделей без додаткового навчання їхніх параметрів. Його ідея полягає в тому, що модель отримує не лише інструкцію до задачі, а й декілька прикладів її правильного виконання. Такі приклади виконують роль демонстрацій, які показують моделі бажаний формат відповіді, логіку міркування або послідовність дій, якої потрібно дотримуватися [10].

У контексті великих мовних моделей few-shot prompting став особливо важливим, оскільки дозволяє адаптувати модель до конкретного типу задачі без проведення повноцінного fine-tuning. Це є суттєвою перевагою, оскільки додаткове навчання моделі потребує значних обчислювальних ресурсів, спеціально підготовлених наборів даних та часу. Натомість few-shot prompting дозволяє впливати на поведінку моделі без зміни її внутрішніх параметрів, лише за рахунок зміни структури вхідного запиту.

Основна ідея методу полягає у включенні до промпта обмеженої кількості прикладів, які демонструють бажаний спосіб розв’язання задачі. Наприклад, у задачах класифікації такі приклади можуть показувати відповідність між вхідними даними та правильним класом. У задачах генерації тексту вони можуть задавати стиль, структуру або рівень деталізації відповіді. У задачах агентної взаємодії приклади можуть

описувати послідовність міркувань, вибір дії та очікуване спостереження після її виконання.

Для LLM-інтегрованих агентів few-shot prompting може використовуватися як спосіб формування локальної політики поведінки. У цьому випадку приклади в промпті показують агенту, як потрібно аналізувати стан середовища, як обирати наступну дію та як реагувати на результат попереднього кроку. Тобто приклади не просто задають формат відповіді, а фактично впливають на стратегію прийняття рішень. Це дозволяє покращити стабільність поведінки агента та зменшити ймовірність некоректних або хаотичних дій.

Водночас few-shot prompting має низку обмежень. По-перше, ефективність цього підходу значною мірою залежить від якості підібраних прикладів. Якщо приклади є нерелевантними, надто загальними або не відповідають поточній задачі, вони можуть не покращити, а навпаки погіршити результат роботи моделі. Тому створення якісного набору демонстрацій є окремою складною задачею, особливо у випадках, коли середовище має великий простір можливих станів.

По-друге, класичний few-shot prompting зазвичай передбачає використання статичного набору прикладів. Такі приклади задаються до початку виконання задачі та не змінюються під час взаємодії агента із середовищем. У динамічних середовищах це створює проблему, оскільки умови виконання задачі можуть змінюватися після кожної дії агента. У результаті приклади, які були корисними на початку, можуть втратити релевантність на наступних етапах виконання.

По-третє, у клієнтоорієнтованих системах задачі користувачів часто мають довільний характер і суттєво відрізняються одна від одної. Неможливо заздалегідь підготувати універсальний набір прикладів, який буде однаково ефективним для всіх можливих сценаріїв. Наприклад, один користувач може вимагати виконання дії у веб-інтерфейсі, інший – аналізу документа, а третій – взаємодії з API або програмним середовищем. Для

кожного такого випадку потрібні різні демонстрації, що ускладнює використання статичного few-shot prompting.

2.1.4 Адаптивний синтез few-shot прикладів

Адаптивний синтез few-shot прикладів можна розглядати як спосіб покращення роботи LLM-агента за рахунок використання попереднього досвіду [11]. На відміну від звичайного few-shot prompting, де приклади заздалегідь задаються вручну і не змінюються під час роботи системи, у цьому підході приклади можуть формуватися динамічно. Тобто система не просто використовує фіксований набір прикладів, а намагається підібрати або сформувати такі приклади, які краще відповідають поточній задачі.

Під час виконання задач агент поступово накопичує історію взаємодії із середовищем. До неї входять попередні дії агента, отримані спостереження, проміжні результати, помилки та фінальний результат виконання. Така історія може бути корисною, оскільки вона показує, які дії були ефективними, а які, навпаки, призводили до помилок або зациклення. На основі цих даних система може виділяти окремі фрагменти досвіду та перетворювати їх у приклади для подальшого використання.

Важливо, що адаптивний синтез не обмежується простим копіюванням уже наявних прикладів. У багатьох випадках поточна задача не буде повністю збігатися з попередніми сценаріями, тому приклади потрібно узагальнювати та адаптувати. Наприклад, система може взяти не конкретну послідовність кліків, а загальний принцип дії: спочатку проаналізувати сторінку, потім знайти потрібний елемент, виконати дію та перевірити результат. Завдяки цьому агент може використовувати попередній досвід навіть у нових ситуаціях.

Синтезовані few-shot приклади зазвичай містять опис задачі, короткий контекст, послідовність дій агента та результат виконання. У такому вигляді вони можуть додаватися до промпта як допоміжні приклади, які підказують

моделі можливий спосіб міркування та виконання задачі. Таким чином, адаптивний синтез few-shot прикладів дозволяє зробити поведінку агента менш статичною і більш залежною від накопиченого досвіду. Це особливо важливо для задач довгогоризонтного планування, де агенту потрібно не лише виконувати окремі дії, а й враховувати попередні кроки, помилки та зміни стану середовища.

2.1.5 Деревоподібний пошук траєкторій

Однією з проблем довгогоризонтного планування є те, що агент не завжди може одразу визначити найкращу наступну дію [12]. У процесі виконання задачі існує кілька можливих напрямів руху, і кожен із них може привести до різних проміжних станів середовища. Це особливо характерно для задач веб-навігації, де після натискання кнопки, переходу за посиланням або введення запиту агент отримує новий контекст, який може як наблизити його до цілі, так і відвести в неправильному напрямку.

Деревоподібний пошук траєкторій спрямований на вирішення цієї проблеми шляхом розгляду кількох альтернативних варіантів дій. У такому підході процес виконання задачі подається як дерево, де вузлами є проміжні стани середовища, а ребрами – дії агента. Замість того щоб одразу виконувати одну дію, агент може сформувати кілька можливих дій-кандидатів, оцінити їх перспективність і продовжити виконання з тієї гілки, яка має найбільшу ймовірність привести до успішного результату.

Базова ідея такого підходу є близькою до методу Tree of Thoughts, у якому мовна модель розглядає не один лінійний ланцюг міркувань, а кілька альтернативних напрямів розв’язання задачі. У випадку LLM-агентів для веб-навігації цей принцип переноситься з простору текстових міркувань у простір реальних станів середовища. Наприклад, один шлях може передбачати використання пошуку на сайті, інший – навігацію через меню,

а ще один – перехід через категорії або фільтри. Кожен із цих шляхів може бути окремо оцінений з погляду наближення до початкової цілі.

Практична реалізація деревоподібного пошуку може включати генерацію кількох можливих дій, виконання або моделювання їхніх наслідків та оцінювання отриманих станів. Після цього система обирає найбільш перспективний напрям і продовжує розгортання саме цієї гілки. Такий підхід дозволяє агенту не тільки реагувати на поточне спостереження, а й частково прогнозувати можливі наслідки різних дій. У задачах веб-навігації це важливо, оскільки користь окремої дії часто стає зрозумілою лише після одного або кількох наступних кроків.

Перевагою деревоподібного пошуку є те, що він зменшує залежність агента від одного локального рішення. Звичайний агент часто обирає дію, яка виглядає найкращою в поточному контексті, однак у довгогоризонтних задачах така дія не завжди є оптимальною для всієї траєкторії. Деревоподібний пошук дозволяє враховувати не лише поточний стан, а й можливі наслідки різних дій. Завдяки цьому агент може краще реагувати на помилки, тупикові ситуації та неочікувані зміни середовища.

Розгляд кількох альтернативних гілок збільшує кількість звернень до мовної моделі та загальну вартість виконання задачі. Крім того, у веб-середовищах не всі дії є зворотними: наприклад, надсилання форми або зміна налаштувань може мати наслідки, які складно скасувати. Тому деревоподібний пошук доцільно розглядати як перспективне розширення для LLM-інтегрованих агентів, яке може посилити ієрархічне планування, пам’ять агента та механізми рефлексії.

2.2 Короткострокова та довгострокова пам’яті в LLM-інтегрованих агентах

У сучасних LLM-інтегрованих агентних системах пам’ять розглядається як один із базових компонентів, поряд із плануванням,

виконанням дій та взаємодією із середовищем. Якщо звичайне використання великої мовної моделі передбачає генерацію відповіді на основі поточного запиту, то агентна система має виконувати послідовність дій, аналізувати результати цих дій, адаптувати план і враховувати попередній досвід.

У загальному вигляді пам’ять можна визначити як процес отримання, збереження, утримання та подальшого відновлення інформації. В оглядах пам’яті для LLM-систем вона також описується як здатність AI-системи зберігати, згадувати та використовувати інформацію з попередніх взаємодій для покращення подальших відповідей і дій. Такий підхід напряму пов’язаний з агентами, оскільки вони виконують складні задачі через планування, використання інструментів, багатоетапне міркування та взаємодію із зовнішнім середовищем.

Найчастіше пам’ять агентів поділяють на короткострокову та довгострокову. Короткострокова пам’ять відповідає за інформацію, яка потрібна під час виконання поточної задачі [13]. Вона містить поточний стан, історію дій, проміжні спостереження, локальні висновки та контекст, необхідний для вибору наступного кроку. Довгострокова пам’ять, навпаки, призначена для збереження інформації, яка може бути корисною не лише в межах одного запуску, а й у майбутніх задачах. У літературі короткострокову пам’ять часто описують як робочу пам’ять усередині поточної задачі, а довгострокову – як пам’ять, що може містити семантичні, процедурні та епізодичні знання.

Використання пам’яті має безпосередній вплив на якість довгогоризонтного планування. Короткострокова пам’ять дозволяє агенту підтримувати послідовність виконання, не втрачати контекст, виявляти повторення дій і коригувати локальні рішення. Довгострокова пам’ять дозволяє використовувати попередній досвід, уникати вже відомих помилок і швидше знаходити правильну стратегію в подібних ситуаціях. Це особливо важливо для веб-навігації, де одна й та сама задача може включати багато

кроків, нестабільні елементи інтерфейсу, помилки середовища та необхідність змінювати план під час виконання.

2.2.1 Збереження історії взаємодії в контексті промпта

Одним із найпростіших способів реалізації пам’яті в LLM-інтегрованих агентних системах є збереження історії взаємодії безпосередньо в контексті промпта. У цьому підході пам’ять агента не існує як окрема структурована база даних або спеціальний модуль, а представлена у вигляді текстового опису попередніх кроків, який додається до кожного наступного запиту мовної моделі. Тобто модель отримує не лише поточну інструкцію користувача, а й інформацію про те, які дії вже були виконані, які спостереження були отримані після цих дій, які проміжні результати були досягнуті та які помилки виникали під час виконання задачі.

У найпростішому вигляді така пам’ять може бути представлена як послідовність записів виду: дія агента, спостереження середовища, результат дії. Наприклад, якщо агент виконує задачу веб-навігації, то після кожного кроку до промпта може додаватися інформація про те, яку кнопку було натиснуто, яка сторінка відкрилася, які елементи інтерфейсу були знайдені, чи наблизила ця дія агента до виконання початкового запиту. Таким чином, мовна модель на наступному кроці приймає рішення не ізольовано, а з урахуванням попереднього контексту виконання.

Перевагою цього способу є його простота. Для реалізації такого підходу не потрібно створювати окрему базу даних, механізм пошуку або складну систему управління знаннями. Уся необхідна інформація передається безпосередньо в мовну модель у вигляді тексту. Це робить підхід зручним для прототипування агентних систем, тестування логіки планування та аналізу поведінки агента на коротких або середніх траєкторіях.

2.2.2 Підсумовування траєкторії виконання

Другим способом реалізації пам’яті в LLM-агентах є підсумовування траєкторії виконання. На відміну від попереднього підходу, де вся історія взаємодії зберігається в контексті промпта майже повністю, у цьому випадку система формує стислий опис того, що вже відбулося під час виконання задачі. Замість передачі всіх дій, спостережень і результатів модель отримує узагальнення попередньої траєкторії, яке містить лише найбільш важливу інформацію для подальшого планування.

Під траєкторією виконання в цьому випадку розуміється послідовність станів, дій та спостережень, які виникали під час роботи агента. У довгогоризонтних задачах така траєкторія може бути досить великою, тому її повне збереження в промпті є неефективним. Підсумовування дозволяє перетворити довгу послідовність кроків у компактний текстовий опис кроків, який зберігає основний зміст виконання задачі.

Перевагою цього підходу є зменшення кількості токенів, необхідних для передачі історії взаємодії. Це особливо важливо для довгогоризонтних задач, де агент може виконувати десятки або сотні кроків. Завдяки підсумовуванню система може підтримувати актуальну пам’ять про попередні події навіть тоді, коли повна історія вже не поміщається в контекстне вікно. Це робить підхід більш масштабованим порівняно з прямим збереженням усієї історії в промпті.

2.2.3 Використання зовнішньої бази знань або векторного сховища

Третім способом реалізації пам’яті в LLM-агентах є використання зовнішньої бази знань або векторного сховища. У цьому випадку пам’ять агента не обмежується контекстним вікном мовної моделі, а зберігається в окремому зовнішньому компоненті [14]. Перед виконанням наступної дії

агент може звертатися до цього сховища, знаходити релевантні записи та додавати їх до промпта як додатковий контекст, щоб на основі цього контекста робити більш відповідні дії.

На відміну від короткострокової пам’яті, яка зазвичай існує лише в межах одного запуску задачі, зовнішня база знань може виконувати роль довгострокової пам’яті. Вона дозволяє зберігати інформацію, отриману під час попередніх запусків системи, типові помилки агента, корисні спостереження про середовище, правила взаємодії з інтерфейсом, приклади успішного виконання задач або узагальнені знання, які можуть бути повторно використані в майбутньому.

Зовнішня пам’ять може бути реалізована у різних формах. Найпростішим варіантом є файлове сховище або ж звичайна база даних, у якій записи зберігаються у вигляді тексту з певними метаданими. Наприклад, кожен запис може містити опис ситуації, тип знання, джерело отримання, час створення та короткий висновок. Іншим варіантом є векторне сховище, де текстові записи перетворюються на векторні представлення за допомогою embedding-моделі. Це дозволяє виконувати семантичний пошук не лише за точним збігом слів, а й за змістовною подібністю.

2.3 Інтерактивні середовища та набори даних для тестування систем

2.3.1 Середовище MiniWobPlusPlus

Середовище MiniWobPlusPlus (рисунок 2.1) є розширенням оригінального середовища MiniWob та використовується для дослідження і тестування підходів до взаємодії агентів із веб-інтерфейсами [15]. Воно містить понад 100 середовищ для веб-взаємодії, а також надає програмні інтерфейси JavaScript і Python. Python-інтерфейс побудований відповідно до

Gymnasium API та використовує Selenium WebDriver для виконання дій у браузері.



Рисунок 2.1 – Приклади задач у середовищі MiniWoB++

Основна особливість MiniWobPlusPlus полягає в тому, що середовище складається з великої кількості невеликих веб-задач, кожна з яких перевіряє окремий тип взаємодії з інтерфейсом. До таких взаємодій належать натискання кнопок, введення тексту, вибір елементів зі списків, робота з чекбоксами, перехід між вкладками, пошук інформації на сторінці, заповнення форм, сортування елементів, взаємодія з календарями та іншими типовими компонентами веб-інтерфейсу. Завдяки цьому MiniWob++ дозволяє досліджувати не лише загальну здатність агента виконувати веб-задачі, а й окремі навички, необхідні для роботи з інтерфейсом.

Задачі в MiniWob++ мають спрощений і контрольований характер. На відміну від реальних веб-сайтів, де сторінки можуть містити велику

кількість нерелевантної інформації, реклами, динамічних елементів та складної структури DOM, задачі MiniWob++ зазвичай представлені у вигляді компактних штучно створених сторінок. Це дозволяє зменшити кількість сторонніх факторів і зосередитися саме на логіці прийняття рішень агентом. Такий підхід є корисним на початкових етапах розробки, коли необхідно перевірити базову працездатність алгоритму, механізму планування або способу взаємодії агента із середовищем.

2.3.2 Середовище WebArena

Середовище WebArena – це автономне, самостійно розгортуване веб-середовище для розробки та тестування автономних агентів [16]. WebArena відтворює вебсайти з п’яти популярних категорій:

− електронна комерція;

− соціальні мережі;

− онлайн-форуми;

− інформаційні бази знань;

− системи керування репозиторіями коду.

У межах WebArena (рисунок 2.2) агент отримує задачу у вигляді природномовної інструкції та повинен самостійно виконати послідовність дій у вебінтерфейсі. Це можуть бути дії, пов’язані з пошуком товарів, роботою з обговореннями, зміною інформації у системі керування контентом або виконанням операцій у середовищі для розробки програмного забезпечення. Через це WebArena є корисним середовищем саме для дослідження довгогоризонтного планування, оскільки виконання задачі зазвичай вимагає не одного кліку, а послідовності взаємопов’язаних кроків.

Важливою особливістю WebArena є те, що середовище не обмежується статичними сторінками. Воно містить функціональні вебзастосунки, які імітують реальні сайти та дозволяють оцінювати не лише

правильність окремої дії, а й кінцевий результат виконання задачі. Крім основних сайтів, WebArena також включає додаткові інструменти та джерела знань, наприклад карту або інструкції користувача, що робить задачі більш схожими на реальні сценарії роботи людини в інтернеті.

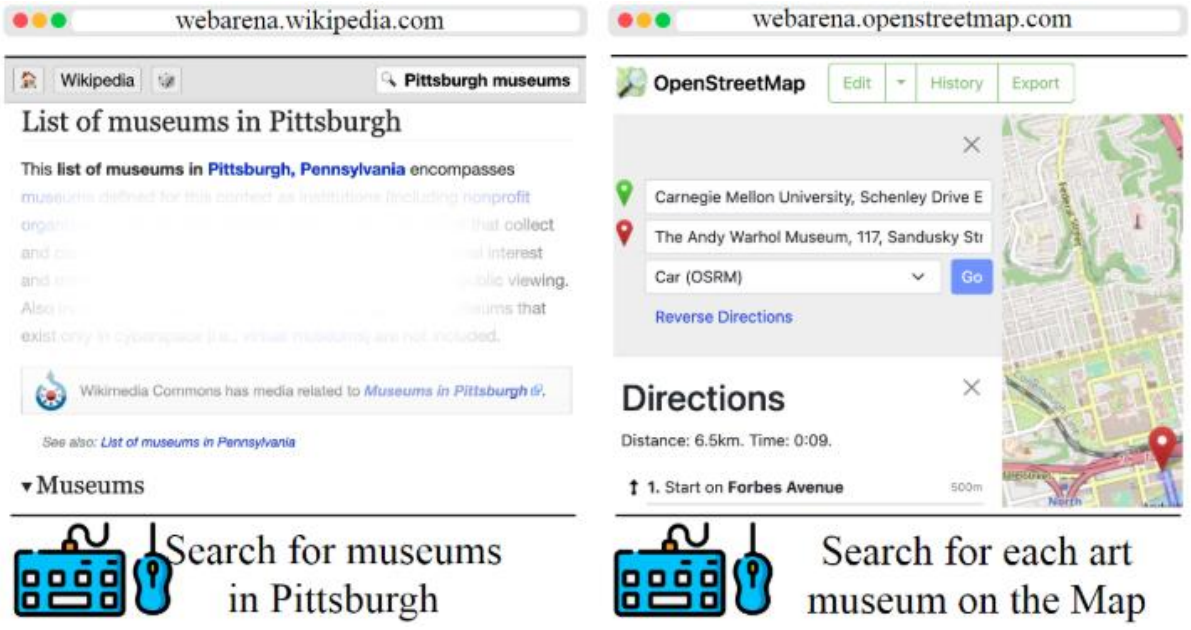


Рисунок 2.2 – Приклади задач у середовищі WebArena

2.3.3 Набір данних Mind2Web

Mind2Web – це набір даних, що був розроблений для розробки, навчання та оцінювання веб-агентів, які виконують задачі у реальних веб-середовищах [17]. На відміну від інтерактивних середовищ, таких як MiniWoB++ або WebArena, Mind2Web не передбачає безпосередньої взаємодії агента з веб-інтерфейсом у режимі реального часу. Натомість він є офлайн-датасетом, у якому вже зафіксовано задачі, стани веб-сторінок та траєкторії виконання, отримані під час взаємодії користувачів із реальними сайтами.

Особливість Mind2Web полягає в тому, що він дозволяє досліджувати поведінку веб-агентів без необхідності розгортання повноцінного

інтерактивного середовища. Це суттєво спрощує процес експериментування, оскільки дослідникові не потрібно налаштовувати локальні копії веб-сайтів, підтримувати їхній стан або враховувати нестабільність зовнішнього середовища.

Набір даних містить понад 2000 задач, зібраних із 137 реальних веб-сайтів. Це робить Mind2Web більш наближеним до реальних умов використання веб-агентів, оскільки задачі не обмежуються штучно створеними інтерфейсами або спрощеними сценаріями. Вони охоплюють різні типи сайтів, структури сторінок, елементи інтерфейсу та способи взаємодії з ними. Завдяки цьому датасет може використовуватися для перевірки здатності агентів узагальнювати поведінку на різні веб-домени.

Кожна задача в Mind2Web супроводжується траєкторією дій, яка демонструє процес її виконання людиною. Такі траєкторії можуть використовуватися як демонстрації для навчання з учителем або як еталонні приклади для оцінювання роботи агентів. Це є важливою перевагою датасету, оскільки агент може навчатися не лише на фінальному результаті, а й на самій послідовності дій, які були виконані для його досягнення. Кожна траєкторія включає:

– тип дії, що була зроблена агентом;

– відповідні елементи DOM, з якими була проведена взаємодія;

– скріншоти веб-інтерфейсу на кожному кроці.

Таким чином, завдяки відсутності потреби у розгортанні локального середовища, відсутності необхідності симуляції веб-інтерфейсів та використанню реальних сайтів, Mind2Web є зручною та не вимогливою альтернативою для прототипування та навчання веб-агентів.

2.4 Метрики оцінювання якості роботи агента

Оцінювання якості роботи LLM-інтегрованого агента є складнішою задачею, ніж оцінювання звичайної моделі машинного навчання, оскільки

результат роботи агента визначається не лише фінальною відповіддю, а й усією траєкторією виконання завдання. Агент послідовно взаємодіє із середовищем, приймає проміжні рішення, виконує дії, отримує спостереження та на їх основі коригує подальшу поведінку. Тому для повноцінного аналізу якості необхідно враховувати не тільки факт успішного завершення задачі, але й ефективність, стабільність та ресурсну вартість процесу виконання.

У межах цієї роботи для оцінювання якості роботи агента розглядаються такі метрики:

− стабільність траекторії;

− відношення успіху;

− довжина траекторії;

− кількість токенів.

Однією з основних метрик є частка успішного виконання завдань. Вона показує, яку частину задач агент зміг виконати повністю та коректно. На відміну від класичних задач класифікації, де результатом є вибір правильного класу, у випадку агентної системи успішність визначається досягненням цільового стану середовища або виконанням усіх вимог початкового запиту користувача. Наприклад, у задачах веб-навігації успішним вважається не просто виконання певної кількості дій, а саме досягнення потрібного результату: знаходження інформації, заповнення форми, перехід до необхідного елемента або завершення заданого сценарію.

Довжина траєкторії характеризує кількість кроків, які агент виконує від початкового стану до завершення задачі. Під кроком у цьому випадку можна розуміти одну дію агента в середовищі: натискання кнопки, введення тексту, перехід на іншу сторінку, вибір елемента інтерфейсу або іншу атомарну операцію. Ця метрика дозволяє оцінити ефективність поведінки агента з погляду кількості дій, необхідних для досягнення результату.

Сама по собі довжина траєкторії не є достатньою метрикою якості, оскільки коротка траєкторія не завжди означає правильне виконання задачі.

Агент може швидко завершити роботу, але не досягти потрібного результату. Тому цю метрику доцільно аналізувати разом із часткою успішного виконання. Найбільш інформативною вона стає при порівнянні різних конфігурацій однієї системи або різних методів планування. Якщо два агенти мають однакову частку успішного виконання, але один із них у середньому використовує менше кроків, його поведінку можна вважати ефективнішою.

Кількість використаних токенів є важливою метрикою для систем, що використовують великі мовні моделі. Оскільки кожен запит до мовної моделі містить системну інструкцію, опис задачі, поточний стан середовища, історію взаємодії та інші допоміжні дані, загальна кількість токенів може швидко зростати під час виконання довгогоризонтних задач. Ця метрика відображає ресурсну вартість роботи агента, оскільки кількість токенів безпосередньо впливає на вартість обчислень, час обробки запитів та обмеження контекстного вікна моделі.

Окреме значення має стабільність траєкторії. У межах цієї роботи стабільність траєкторії розглядається як здатність агента послідовно просуватися до цілі без повторення однакових або неефективних дій [18]. Нестабільна поведінка може проявлятися у вигляді циклів, повернення до вже відвіданих станів, повторного виконання дій, які раніше не дали результату, або тривалого перебування в одній і тій самій частині середовища без реального прогресу.

3 ПРАКТИЧНІ ДОСЛІДЖЕННЯ

3.1 Реалізовані агенти

3.1.1 Планувальник глобального плану

Задачею планувальника глобального плану є декомпозиція високорівневого запиту користувача на послідовність атомарних підзадач. Цей етап є частиною механізму динамічного формування запитів для подальших агентів системи. Окрім цього, планувальник глобального плану покращує процес виконання задачі за рахунок поділу початкового користувацького запиту на менші та більш конкретні підзадачі.

На вхід глобальний планувальник отримує текстове формулювання задачі користувача, а також зображення поточного стану вебсторінки. Використання зображення потрібне для базового розуміння контексту задачі, яку необхідно виконати. Це важливо, оскільки заздалегідь не завжди відомо, на якому саме вебресурсі буде виконуватися задача, а текстовий запит користувача може не містити всієї необхідної інформації.

Важливо зазначити, що глобальний план, згенерований агентом, не містить низькорівневих дій і не має прямої прив’язки до конкретних елементів на сторінці. Тобто на цьому етапі не визначаються конкретні кнопки, поля введення або інші елементи інтерфейсу, з якими має взаємодіяти агент. Такий підхід дає можливість краще контролювати якість сформованого плану та відокремити високорівневе планування від низькорівневого виконання задачі. Це є важливим для задач довгогоризонтного планування, оскільки початкова задача користувача зазвичай потребує не однієї дії, а послідовності кроків.

У межах реалізації були додані гіперпараметри для посилення контролю над вихідними даними агента. Зокрема, було введено обмеження на мінімальну та максимальну кількість згенерованих підзадач. Це дозволяє

контролювати обсяг сформованого плану, зменшити надмірне використання токенів і зробити поведінку моделі більш детермінованою.

3.1.2 Агент спостереження та рефлексії

Задачами агента спостереження та рефлексії є контроль актуальності глобального плану, аналіз поточної траєкторії виконання та визначення потреби в отриманні додаткових знань. У межах запропонованої системи цей агент виконує роль рефлексивного компонента, який підтримує загальний напрям виконання користувацької задачі. Це є важливим для задач довгогоризонтного планування, оскільки під час виконання окремих локальних дій агент може відхилятися від початкової мети або потребувати уточнення подальшого курсу. Фактично агент спостереження та рефлексії аналізує, наскільки поточні дії відповідають глобальному плану, формує необхідний контекст для агента виконання дій і частково виконує функцію базового оркестратора в циклі розв’язання задачі.

На вхід агент отримує користувацьке завдання, глобальний план, поточну траєкторію та зображення поточної вебсторінки. Результатом роботи агента є логічне значення чи було виконане завдання, що сигналізує про вихід з циклу виконання завдання, словесний опис поточного зображення, словесний опис результату попередньої дії агента, словесна підказка агенту виконання дій, що має вказати подальший напрямок виконання завдання.

Під час роботи агент спостереження та рефлексії оцінює, чи відповідає поточний стан середовища початковій задачі користувача та глобальному плану. Для цього він аналізує не лише останнє спостереження, а й попередню дію агента та її результат. Тобто агент визначає, чи наблизила виконана дія систему до розв’язання задачі, чи навпаки відвела її від потрібного напряму.

Також агент перевіряє відповідність поточного стану глобальному плану. Якщо під час виконання задачі виявляється, що сформований раніше план уже не відповідає ситуації на сторінці або початковому запиту користувача, агент може визначити потребу в його уточненні. Окремо аналізується, чи достатньо поточного контексту для подальшого виконання задачі, або чи потрібно звернутися до сховища знань для отримання додаткової інформації.

Крім цього, агент відповідає за визначення стану виконання задачі. Він перевіряє, чи була задача вже вирішена, чи система потрапила у тупиковий стан, а також чи не виникло циклічне повторення однакових або схожих дій.

Агент спостереження та рефлексії є важливою частиною мультиагентної системи під час виконання задач довгогоризонтного планування. Завдяки відокремленю процесу прийняття рішень і аналізу ситуації від безпосереднього виконання дій у вебсередовищі, система стає більш стійкою до помилок під час вебнавігації. Крім цього, розділення механізмів дії та рефлексії спрощує розробку системи, оскільки кожен агент відповідає за більш конкретну частину загального процесу.

3.1.3 Агент виконання дій

Задачею агента виконання дій є безпосередня навігація вебресурсом через взаємодію з середовищем. Фактично цей агент найбільше прив’язаний до конкретного способу комунікації з вебсередовищем, оскільки саме він перетворює сформоване рішення системи на конкретну дію. У звичайному браузері навігація розрахована переважно на людину та виконується за допомогою миші, клавіатури або їх комбінації: натискання кнопок, введення тексту, прокручування сторінки, вибір елементів зі списків тощо.

У розглянутих тестових середовищах така взаємодія реалізується через спеціальні бібліотеки, надані розробниками цих середовищ. Вони

виконують роль посередника між браузерним інтерфейсом і програмним кодом. Тому агент виконання дій не працює з вебсторінкою напряму як користувач, а використовує доступний функціонал клієнтської бібліотеки для виконання дій у середовищі. Наприклад, залежно від середовища це може бути натискання на елемент за його ідентифікатором, введення тексту в поле, прокручування сторінки або виконання клавіатурної команди.

На вхід агент отримує користувацьке завдання, зображення поточної вебсторінки, текстову анотацію доступних елементів інтерфейсу, а також підказку від агента спостереження та рефлексії. Саме ця підказка використовується як основний короткостроковий орієнтир. Водночас агент додатково перевіряє її відповідність поточному зображенню та анотації сторінки.

Результатом роботи агента є обрана дію та коротке пояснення причини її вибору. Дія має відповідати підтримуваному формату середовища, наприклад: натискання на елемент за його ідентифікатором, введення тексту, прокручування сторінки, натискання клавіші або завершення задачі з фінальною відповіддю.

На відміну від агента глобального планування та агента спостереження і рефлексії, агент виконання дій напряму пов’язаний з обраною клієнтською бібліотекою для комунікації з вебсередовищем. Через це він має меншу придатність до повторного використання, оскільки його логіка залежить від конкретного формату дій, який підтримує певне середовище або бібліотека. Водночас відокремлення агента виконання дій від інших агентів дозволяє змінювати лише цю частину системи у разі переходу на іншу бібліотеку або інший спосіб взаємодії з вебресурсом. Тобто глобальне планування, рефлексія та загальна логіка роботи системи можуть залишатися майже незмінними, а адаптація виконується переважно на рівні агента виконання дій. Це підвищує придатність до повторного використання не окремого агента, а самого підходу загалом.

3.1.4 Менеджер знань

Задача агента з менеджменту знань аналіз траекторій за результатам виконання завадань задля акумулювання досвіду, отриманого під час виконання задач, що можуть бути перевикористанні в майбутніх запусках.

Менеджер знань запускається наприкінці циклу виконання задачі. На вхід він отримує користувацьке завдання, глобальний план, повну траєкторію дій, статус завершення, фінальний результат та наявну базу знань. На основі цих даних агент оцінює, чи виникали під час виконання повторювані помилки, цикли, особливості інтерфейсу або інші спостереження, які можуть бути повторно використані.

У системі використовується кілька типів знань: інформаційні записи, записи про помилки та спостереження. Інформаційні записи містять загальні правила, інструкції або корисні підказки для виконання задач. Записи про помилки описують проблемні ситуації, невдалі дії, циклічну поведінку або тупикові стани. Спостереження, у свою чергу, фіксують корисні особливості поведінки вебсторінки або тестового середовища, які можуть бути використані під час наступних запусків системи.

Такий підхід дозволяє відокремити механізм накопичення досвіду від агента спостереження та рефлексії. Агент спостереження та рефлексії зосереджується на поточному виконанні задачі, аналізі стану середовища та підтриманні правильного напряму виконання. Менеджер знань, навпаки, працює вже після завершення задачі та аналізує отриману траєкторію більш узагальнено. Це робить архітектуру системи більш зрозумілою, контрольованою та зручною для налагодження. Окрім цього, така структура дозволяє користувачу самостійно додавати знання до бази даних, якщо певні правила або спостереження вже відомі заздалегідь.

Таким чином, менеджер знань виконує роль довготривалої пам’яті системи. Він дозволяє накопичувати практичний досвід взаємодії з

вебсередовищем, зберігати його у структурованому вигляді та використовувати для покращення майбутніх виконань.

3.2 Процес виконання задачі

У межах реалізованої системи процес виконання задачі організовано як послідовність етапів, що включають ініціалізацію середовища, формування глобального плану, ітеративне виконання дій, перевірку результату та завершальний аналіз траєкторії (рисунок 3.1).

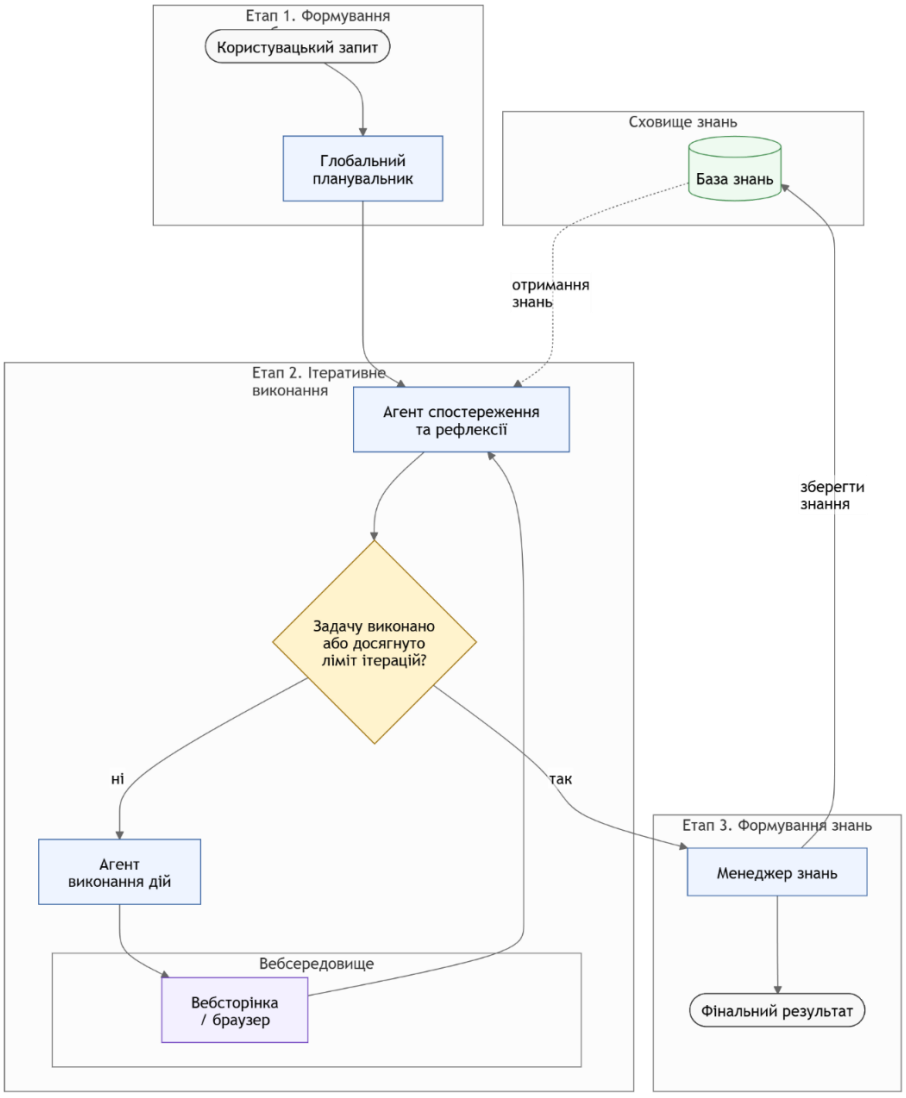


Рисунок 3.1 – Діаграма мультиагентнаої системи

На початку виконання система отримує користувацький запит, який описує кінцеву мету задачі. Після цього ініціалізується вебсередовище, з якого отримується початкове спостереження. Це спостереження може містити текстовий опис доступних елементів сторінки, зображення поточного стану вебінтерфейсу або інші дані, необхідні для початкового аналізу. Отримана інформація використовується для формування початкового глобального плану.

Після формування глобального плану система переходить до ітеративного циклу виконання. На кожній ітерації спочатку аналізується поточний стан задачі. Система перевіряє, чи відповідає поточний стан початковому запиту користувача, чи зберігається актуальність глобального плану, чи була попередня дія корисною та чи не досягнуто вже фінального результату. Якщо на цьому етапі визначається, що задача виконана, система завершує роботу та повертає знайдений результат.

Якщо задача ще не виконана, формується наступна локальна дія. Для цього використовується поточне спостереження середовища, глобальний план, історія попередніх дій та додатковий контекст, отриманий під час аналізу прогресу.

Після формування локальної дії вона передається до середовища для виконання. Середовище змінює свій стан і повертає нове спостереження. Таким чином, кожна ітерація залежить від результату попереднього кроку.

Під час виконання також зберігається траєкторія роботи системи. До неї входять виконані дії, короткий опис спостережень, причини вибору дій, а також можливі помилки або відхилення від очікуваного перебігу задачі.

Окремо в алгоритмі передбачено обмеження на максимальну кількість ітерацій. Це потрібно для того, щоб система не продовжувала виконання нескінченно у випадку, якщо задача не може бути розв’язана, середовище змінилося неочікуваним способом або агент почав повторювати однакові дії. Якщо ліміт ітерацій досягнуто, виконання завершується зі статусом невдалого або неповного виконання.

Після завершення виконання, незалежно від результату, траєкторія передається на завершальний аналіз. На цьому етапі система оцінює весь шлях виконання задачі: початковий запит, сформований глобальний план, послідовність локальних дій, проміжні спостереження, помилки та фінальний статус. На основі цього можуть бути сформовані знання для подальшого використання в інших запусках системи.

3.3 Технічні особливості обраних тестових середовищ

3.3.1 Особливості середовища WebArena

У межах експериментів середовище WebArena використовується як реалістична симуляція вебсайту. Саме середовище містить декілька симульованих вебресурсів різного типу, однак з урахуванням вимог до обчислювальних ресурсів і можливостей локального середовища виконання було прийнято рішення зосередитися лише на вебсторінці магазину брендового одягу.

Для проведення експериментів відповідний вебдодаток було розгорнуто в контейнеризованому середовищі Docker. Клієнтський код WebArena було завантажено та налаштовано в локальному середовищі виконання.

Після цього було виконано підготовку середовища та сформовано набір тестових випадків, які використовувалися для перевірки роботи реалізованої мультиагентної системи.

Для проведення експериментів використовувався режим, у якому можна візуально спостерігати за діями агента. Тобто браузер відкривався у видимому режимі, а виконання дій було додатково сповільнено. Це було зроблено для зручності налагодження, оскільки під час тестування можна було бачити, які саме дії виконує агент, як змінюється сторінка після кожного кроку та на якому етапі можуть виникати помилки.

Однією з основних можливостей WebArena є отримання текстової анотації поточного стану сторінки. У такому режимі агент отримує не HTML-код напряму, а текстове представлення дерева доступності. У цьому представленні елементи вебсторінки описуються через числові ідентифікатори, ролі, назви та додаткові стани. Водночас у межах експериментів було виявлено, що дані текстової анотації не завжди достатньо для стабільної роботи з окремими елементами інтерфейсу. Це особливо стосується випадаючих списків, переліків та деяких клікабельних елементів, які можуть бути неповно або неоднозначно представлені в анотації.

Взаємодія агента із середовищем відбувається через текстові команди, що наведені у лістингу 3.1, встановленого формату. На кожній ітерації локальний агент формує одну дію, після чого вона передається до WebArena для виконання. Якщо дія пов’язана з конкретним елементом сторінки, агент повинен вказати ідентифікатор цього елемента. Після виконання дії середовище повертає оновлений стан сторінки, який використовується на наступному кроці.

Лістинг 3.1 – Простір можливих дій у середовищі WebArena

```
Valid action formats. Do not remove [ or ] characters, they
are mandatory.
- click [element_id]
- hover [element_id]
- type [element_id] [<text>] [enter_flag] / [0] or [1]
- press [key_combination]
- scroll [up|down]
- goto [url]
- new_tab
- tab_focus [page_number]
- close_tab
- stop [answer]
```

3.3.2 Особливості середовища MiniWob

У межах експериментів середовище MiniWobPlusPlus використовується як мінімалістичне середовище виконання для перевірки роботи агента. Воно складається з набору експериментальних вебзадач, змодельованих через бібліотеку gymnasium. Взаємодія з інтерфейсом відбувається через клієнтську бібліотеку, що очікує текстові команду у форматі, що наведені у лістингу 3.2, яка дозволяє агенту отримувати стан середовища, виконувати дії та отримувати результат після кожного кроку.

Більшість задач MiniWobPlusPlus реалізовані як окремі вебсторінки з простим інтерфейсом. При цьому для виконання задачі зазвичай потрібно більше ніж один крок, наприклад знайти потрібний елемент, натиснути на нього, вибрати значення зі списку або підтвердити результат. Через це MiniWobPlusPlus можна розглядати як спрощене середовище для перевірки задач довгогоризонтного планування. Додатковою перевагою є те, що для його використання не потрібно окремо розгортати вебдодаток у Docker або іншому середовищі хостингу.

На відміну від WebArena, MiniWobPlusPlus не надає готову текстову анотацію сторінки у вигляді дерева доступності. Замість цього агент отримує сирий DOM-об’єкт, який містить структуровану інформацію про елементи сторінки. Це дає більше технічних даних, але водночас може збільшувати кількість токенів, оскільки DOM часто містить зайву для конкретної задачі інформацію. Крім цього, середовище надає зображення поточного стану сторінки, яке використовується як додатковий візуальний контекст у випадках, коли одного DOM-представлення недостатньо.

Лістинг 3.2 – Простір можливих дій у середовищі MiniWobPlusPlus

```
# No-op
NONE = "NONE"
# Mouse actions with coordinates
```

Продовження лістингу 3.2

```
MOVE_COORDS = "MOVE_COORDS"
CLICK_COORDS = "CLICK_COORDS"
DBLCLICK_COORDS = "DBLCLICK_COORDS"
MOUSEDOWN_COORDS = "MOUSEDOWN_COORDS"
MOUSEUP_COORDS = "MOUSEUP_COORDS"
# Mouse wheel
SCROLL_UP_COORDS = "SCROLL_UP_COORDS"
SCROLL_DOWN_COORDS = "SCROLL_DOWN_COORDS"
# Mouse actions with elements
CLICK_ELEMENT = "CLICK_ELEMENT"
# Keyboard
PRESS_KEY = "PRESS_KEY"
TYPE_TEXT = "TYPE_TEXT"
TYPE_FIELD = "TYPE_FIELD"
FOCUS_ELEMENT_AND_TYPE_TEXT = "FOCUS_ELEMENT_AND_TYPE_TEXT"
FOCUS_ELEMENT_AND_TYPE_FIELD = "FOCUS_ELEMENT_AND_TYPE_FIELD"
```

3.4. Проведені експерименти

3.4.1 Проведення експериментів на основі MiniWobPlusPlus

Для експериментальної перевірки та вирахування метрик запропонованого методу було використано середовище MiniWobPlusPlus, яке дозволяє оцінити роботу агента в задачах взаємодії з веб-інтерфейсом у контрольованих умовах.

У межах експериментів було порівняно дві конфігурації агента: базову модель та запропоновану модель, що використовує динамічне планування й адаптивне формування політики дій. Для кожної конфігурації було проведено по 30 експериментів, що дозволило отримати порівнювані результати та проаналізувати відмінності у поведінці агентів.

Оцінювання результатів виконувалося за такими метриками: частка успішно виконаних задач, довжина траєкторії, кількість використаних токенів та стабільність траєкторії. Результати екпериментів зазначені у таблиці 3.1.

Таблиця 3.1– Результати експериментів

Метрика	Базова Модель	Запропонована модель
Частка успіху	0.8	0.9
Частка циклів	0.34	0.13
Медіана токенів	5844	7870
Медіана довжини траєкторії	2	2

У результаті проведених експериментів можна зробити висновок, що запропонована модель показала трохи кращі результати порівняно з базовою моделлю. Зокрема, частка успішного виконання задач у запропонованої моделі була вищою. Це означає, що використання динамічного планування та адаптивного формування політики дій допомогло агенту частіше досягати потрібного результату в середовищі MiniWobPlusPlus.

Також запропонована модель показала вищу стабільність траєкторії. Це свідчить про те, що агент рідше повторював однакові дії або потрапляв у цикли під час виконання задач. Окремо можна зазначити, що у покращеній моделі майже не було випадків порушення максимальної кількості ітерацій. Тобто агент рідше доходив до ситуації, коли задача завершується не через досягнення цілі, а через перевищення дозволеної кількості кроків.

Водночас запропонована модель у середньому використовувала більше токенів, ніж базова. Це можна пояснити тим, що вона виконує більше проміжних дій, пов’язаних з аналізом стану, плануванням і вибором наступного кроку. Медіанна довжина траєкторії в обох моделей загалом була приблизно однаковою. Це пов’язано з тим, що більшість задач у MiniWobPlusPlus є досить простими і зазвичай потребують невеликої кількості дій для виконання, що є невеликим обмеженням середовища MiniWobPlusPlus.

3.4.2 Спостереження під час виконання WebArena

Під час виконання експериментів у середовищі WebArena основний акцент був зроблений на спостереженні за поведінкою системи під час виконання задач.

Під час тестування було помічено, що агент іноді починає повторювати одні й ті самі дії. Наприклад, він може кілька разів натискати на один і той самий елемент, повертатися до вже пройденого стану або знову намагатися виконати дію, яка вже не спрацювала. Також були ситуації, коли агент фактично заходив у глухий кут. У деяких випадках механізм рефлексії допомагав йому вийти з такого стану. Наприклад, якщо після виконання дії стан сторінки майже не змінювався, агент міг зробити висновок, що попередній крок був невдалим, і спробувати інший варіант дії.

Однією з найбільших проблем стали випадні списки. Через особливості середовища неможливо взаємодіяти з випадними списками через операцію натискання мишею. На відміно від того, як навігація списком виконується людиною, для системи будь-яка дія з випадним списком стала складною перешкодою. Навігація випадним списком може відбуватися лише, чрез через клавіатуру: спочатку активувати список, потім використовувати кнопку стрілка верх або кнопку стрілка вниз для переходу між варіантами, а після цього натиснути кнопку виконання для вибору.

Отже, під час виконання WebArena було помічено, що проблеми виникають не тільки через складність довгогоризонтного планування, а й через технічні особливості взаємодії з інтерфейсом. Рефлексія в деяких випадках допомагала виправити помилки та вийти з тупикових ситуацій, але вона не завжди могла вирішити проблеми, пов’язані зі складними або нестабільними елементами інтерфейсу.

3.5 Можливі напрями вдосконалення

Існує декілька нереалізованих напрямів подальшого розвитку моделі, однак через обмеження за часом, вартістю експериментів та доступними обчислювальними ресурсами вони не були реалізовані в межах цієї роботи. До таких покращень можна віднести використання векторної бази даних для сховища знань агента та додавання механізму репланування глобального плану.

3.5.1 Векторна база даних

Одним із можливих покращень було використання векторної бази даних для реалізації сховища знань. Ідея полягала в тому, щоб зберігати попередній досвід агента, приклади виконання задач, типові помилки та корисні стратегії не просто у вигляді тексту, а у вигляді записів, які можна шукати за семантичною схожістю.

У такому варіанті для кожної задачі або ситуації формувався б вектор представлення. Самі знання б зберігалися б як додаткова інформація, пов’язана з цим вектором. Коли агент отримує нову задачу, для неї також формується вектор, після чого система шукає найближчі за змістом записи у векторній базі даних. Таким чином, отримання знаннь мало б більше контрольовану форму, через те що отримання знань відбувалось б на базі семантичного пошуку, а не рішенням моделі.

Однак у межах цієї роботи цей підхід не був реалізований. Основна проблема полягає в тому, що для повноцінного використання векторної бази даних потрібно не лише окремо розгортати саму базу даних, але й забезпечити формування векторних представлень для задач і записів сховища знань. Локальне розгортання моделі для створення векторних представлень потребує значних обчислювальних ресурсів, що в межах цієї роботи було нереалістичним. Використання хмарних сервісів для формування таких представлень також є можливим варіантом, однак воно потребує додаткових фінансових витрат, особливо при великій кількості експериментів. Тому через обмеження за часом, вартістю та доступними ресурсами це покращення не було включене до фінальної експериментальної частини.

3.5.2 Динамічна регенерація глобального плану

Ще одним можливим покращенням є динамічне репланування глобального плану. У поточному варіанті глобальний план формується на початку виконання задачі та використовується як основна стратегія для подальших дій агента. Це робить поведінку агента більш контрольованою, тому що він не змінює загальний напрям після кожного кроку.

Водночас у деяких довгогоризонтних задачах повна відсутність репланування може бути проблемою. Початковий план не завжди може врахувати всі можливі ситуації, які виникають під час виконання задачі. Агент може зіткнутися з неочікуваним станом середовища, помилкою виконання або ситуацією, коли попередній план уже не відповідає поточному стану. У таких випадках було б корисно мати механізм перегляду або оновлення глобального плану.

Основна складність полягає в тому, що репланування робить систему більш динамічною, але при цьому може зменшити контроль над поведінкою агента. Якщо агент буде занадто часто змінювати глобальний план, це може

призвести до нестабільності, втрати початкової цілі або збільшення кількості використаних токенів. Тобто репланування може покращити адаптивність, але водночас зробити поведінку менш передбачуваною.

3.6 Порівняння з іншими моделями

Для порівняння з розробленим підходом було обрано дві агентні системи, що демонструють одні з найвищих результатів у задачах веб- навігації на бенчмарку WebArena: OpAgent та ColorBrowserAgent. Вибір цих підходів зумовлений тим, що обидві системи орієнтовані на виконання складних довгогоризонтних задач у браузерному середовищі, де агент повинен не лише виконувати окремі дії, а й підтримувати послідовність взаємопов’язаних кроків, адаптуватися до змін стану сторінки та відновлюватися після помилок.

Водночас ці підходи розв’язують проблему довгогоризонтної веб-навігації різними способами. OpAgent робить акцент на модульній архітектурі, у якій окремі компоненти відповідають за планування, прив’язку дій до елементів інтерфейсу, рефлексію та підсумовування траєкторії виконання. Такий підхід дозволяє покращити взаємодію з елементами сторінки та підвищити стійкість агента до помилок під час виконання задачі. ColorBrowserAgent, навпаки, більше зосереджується на накопиченні та використанні знань: система застосовує механізми адаптації знань на основі зворотного зв’язку та прогресивного підсумовування, що дає змогу зменшити нестабільність під час довгих послідовностей дій.

OpAgent має архітектуру Planner–Grounder–Reflector–Summary. Reflector перевіряє, чи була попередня дія корисною і чи не завершена вже задача [19]. Planner на основі цього формує наступну інструкцію, а Summary використовується для фінального узагальнення результату.

Головна особливіть OpAgent полягає у наявності окремого модуля Grounder, який відповідає за перетворення інструкції у конкретну дію на

сторінці [20]. Для цього в OpAgent використовується окремо донавчена VLM-модель. Через це OpAgent може бути точнішим при натисканні на конкретні елементи інтерфейсу, але водночас такий підхід достатньо комплексний, оскільки потребує окремого навчання моделі.

ColorBrowserAgent базується на елементі Adaptive Knowledge Base – база знань, у якій зберігаються правила та підказки щодо роботи з різними сайтами. Якщо агент помиляється, людина-експерт може проаналізувати помилку та додати до бази знань узагальнене правило. Це правило потім може використовуватися не тільки для однієї конкретної задачі, а й для інших схожих ситуацій.

Ще однією важливою особливістю ColorBrowserAgent є knowledge-aligned progressive summarization. Цей механізм стискає історію взаємодії, але не просто скорочує її, а намагається зберегти ті частини, які відповідають релевантним знанням і початковій цілі задачі. Це важливо для довгих задач, тому що агент може втратити контекст після великої кількості кроків.

ВИСНОВКИ

Протягом виконання даної роботи було проведено теоретичне дослідження задач довгогоризонтного планування для LLM-інтегрованих агентів. Було проаналізовано сучасний стан розвитку великих мовних моделей та агентних систем на їх основі, а також розглянуто особливості застосування таких систем в інтерактивних середовищах.

Було проведено аналіз існуючих підходів до динамічного планування, зокрема ієрархічного планування, адаптивного синтезу навичок, використання механізмів пам’яті та рефлексії. У межах аналізу було визначено, що задачі довгогоризонтного планування мають значну залежність від середовища виконання, оскільки агент повинен не лише формувати послідовність дій, а й постійно враховувати зміну стану середовища, результати попередніх кроків та можливі обмеження інтерфейсу.

Також було проаналізовано існуючі середовища та набори даних для оцінювання роботи агентів, а саме MiniWoB++, WebArena та Mind2Web. Було визначено їх придатність для різних етапів дослідження: прототипування, офлайн-аналізу та фінального оцінювання. Окрім цього, було розглянуто основні метрики оцінювання ефективності агентів, серед яких відношення успіху, довжина траєкторії, кількість токенів та стабільність траєкторії.

У межах практичної частини було реалізовано підхід до динамічного планування для LLM-інтегрованого агента, який поєднує формування плану, покрокове виконання дій, аналіз поточного стану та рефлексію над результатами виконання. Під час експериментів було виявлено, що однією з основних проблем у задачах веб-навігації є виникнення циклів, тупикових ситуацій та помилок, пов’язаних з особливостями інтерфейсу середовища. Зокрема, агент міг повторювати однакові або подібні дії, не досягати

потрібного стану сторінки або неправильно інтерпретувати доступні елементи керування.

Отримані результати показали, що задачі довгогоризонтного планування у веб-середовищах значною мірою залежать не лише від якості мовної моделі, а й від способу подання стану середовища, доступного простору дій та стабільності взаємодії з елементами інтерфейсу. Через це навіть правильно сформований план не завжди гарантує успішне виконання задачі, оскільки на окремих кроках можуть виникати помилки взаємодії, неточності в інтерпретації сторінки або ситуації, в яких агент не може перейти до наступного логічного етапу.

Окремо було розглянуто роль рефлексії під час виконання задач. Було встановлено, що рефлексія може допомагати агенту виявляти повторення дій, аналізувати неефективні кроки та змінювати подальшу стратегію виконання. У деяких випадках це дозволяло виходити з циклів або тупикових ситуацій, оскільки агент отримував можливість оцінити не лише поточний стан, а й попередню траєкторію виконання. Водночас рефлексія не усуває всі проблеми повністю, оскільки її ефективність також залежить від повноти та точності інформації, яку агент отримує із середовища.

На основі пройдених траєкторій було сформовано декілька одиниць знань, які можуть використовуватися під час подальших запусків системи. Такі знання можуть містити інформацію про успішні стратегії, типові помилки, особливості поведінки окремих елементів вебінтерфейсу або ситуації, в яких агент схильний до зациклення. Це дозволяє розглядати накопичений досвід не лише як журнал виконання, а як основу для подальшого вдосконалення поведінки агента.

У процесі дослідження було показано, що для задач довгогоризонтної веб-навігації важливими є не лише планування та генерація дій, а й механізми аналізу траєкторії, виявлення помилок, виходу з циклів, роботи з тупиковими ситуаціями та накопичення знань на основі попереднього досвіду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ReAct: synergizing reasoning and acting in language models / S. Yao et al. *arXiv*. 2022. URL: https://doi.org/10.48550/arXiv.2210.03629 (date of access: 01.04.2026).

2. Reflexion: language agents with verbal reinforcement learning / N. Shinn et al. *arXiv*. 2023. URL: https://doi.org/10.48550/arXiv.2303.11366 (date of access: 01.04.2026).

3. AutoGen: enabling next-gen LLM applications via multi-agent conversation / Q. Wu et al. *arXiv*. 2023. URL: https://doi.org/10.48550/arXiv.2308.08155 (date of access: 01.04.2026).

4. Plan-and-Act: improving planning of agents for long-horizon tasks / L. Erdogan et al. *arXiv*. 2025. URL: https://doi.org/10.48550/arXiv.2503.09572 (date of access: 01.04.2026).

5. Why reasoning fails to plan: a planning-centric analysis of long-horizon decision making in LLM agents / Z. Wang et al. *arXiv*. 2026. URL: https://doi.org/10.48550/arXiv.2601.22311 (date of access: 01.04.2026).

6. BrowserAgent: building web agents with human-inspired web browsing actions / T. Yu et al. *arXiv*. 2025. URL: https://doi.org/10.48550/arXiv.2510.10666 (date of access: 01.05.2026).

7. KLong: training LLM agent for extremely long-horizon tasks / Y. Liu et al. *arXiv*. 2026. URL: https://doi.org/10.48550/arXiv.2602.17547 (date of access: 01.04.2026).

8. HiPlan: hierarchical planning for LLM-based agents with adaptive global-local guidance / Z. Li et al. *arXiv*. 2025. URL: https://doi.org/10.48550/arXiv.2508.19076 (date of access: 01.04.2026).

9. Cooperative multi-agent planning with adaptive skill synthesis / Z. Li et al. *arXiv*. 2025. URL: https://doi.org/10.48550/arXiv.2502.10148 (date of access: 01.04.2026).

10. Language models are few-shot learners / T. Brown et al. *arXiv*. 2020. URL: https://doi.org/10.48550/arXiv.2005.14165 (date of access: 01.04.2026).

11. Махотіло В.К., наук.керівник Шевченко О.Ю. Формування few-shot прикладів на основі історії дій агента. 30-ий Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». – Харків: ХНУРЕ, 22 – 24 квітня 2026. С. 72 – 73.

12. ExpeL: LLM agents are experiential learners / A. Zhao et al. *arXiv*. 2023. URL: https://doi.org/10.48550/arXiv.2308.10144 (date of access: 01.05.2026).

13. MemGPT: towards LLMs as operating systems / C. Packer et al. *arXiv*. 2023. URL: https://doi.org/10.48550/arXiv.2310.08560 (date of access: 01.04.2026).

14. Retrieval-augmented generation for knowledge-intensive NLP tasks / P. Lewis et al. *arXiv*. 2020. URL: https://doi.org/10.48550/arXiv.2005.11401 (date of access: 01.05.2026).

15. MiniWoB++ documentation / *The Farama Foundation*. 2024. URL: https://miniwob.farama.org/index.html (date of access: 01.04.2026).

16. Mind2Web: towards a generalist agent for the web / X. Deng et al. *arXiv*. 2023. URL: https://doi.org/10.48550/arXiv.2306.06070 (date of access: 01.04.2026).

17. AgentRewardBench: evaluating automatic evaluations of web agent trajectories / X. Lù et al. *arXiv*. 2025. URL: https://doi.org/10.48550/arXiv.2504.08942 (date of access: 01.04.2026).

18. OpAgent: operator agent for web navigation / Y. Guo et al. *arXiv*. 2026. URL: https://doi.org/10.48550/arXiv.2602.13559 (date of access: 01.05.2026).

19. ColorBrowserAgent: complex long-horizon browser agent with adaptive knowledge evolution / J. Wang et al. *arXiv*. 2026. URL: https://doi.org/10.48550/arXiv.2601.07262 (date of access: 01.05.2026).